IBM Toolbox per Java

Indice

IBM Toolbox per Java 1
Novità in V5R3 2
Stampa di questo argomento 4
Installazione e gestione di IBM Toolbox per Java . . 5
Gestione dell'installazione di IBM Toolbox per Java 5
Installazione di IBM Toolbox per Java 6
Proprietà di sistema 16
Classi IBM Toolbox per Java 23
Classi Access 23
Classi Commtrace 178
Classi HTML 189
Classi ReportWriter 220
Classi Resource 222
Classi Security 225
Classi servlet 236
Classi Utility 243
Classi Vaccess 255
Graphical Toolbox 298
Configurazione del Graphical Toolbox 304
Creazione dell'interfaccia utente 306
Visualizzazione dei pannelli al tempo di esecuzione 309
Modifica dei documenti di aiuto creati dal GUI Builder 313
Utilizzo del Graphical Toolbox in un browser 317
Barra degli strumenti Programma di creazione pannello GUI Builder 321
Bean IBM Toolbox per Java 323
JDBC 324
Aggiornamenti del supporto JDBC di IBM Toolbox per Java 325
Funzioni potenziate di JDBC per OS/400 Versione 5 Release 2 326
Proprietà JDBC di IBM Toolbox per Java . . . 328
Tipi SQL JDBC 345
Supporto proxy 346
Descrizione lunga della Figura 1: come un client standard e un client proxy si collegano ad un server (rzahh505.gif) 350
Descrizione lunga della Figura 1: confronto della dimensione dei file jar proxy rispetto ai file jar standard (rzahh502.gif) 350
Esempio: esecuzione di un'applicazione Java utilizzando il supporto Proxy 351
Esempio: esecuzione di un'applet Java utilizzando un supporto proxy 351
Esempio: esecuzione di un'applicazione Java utilizzando il supporto Tunneling Proxy . . . 352
SSL (Secure Sockets Layer) e JSSE (Java Secure Socket Extension) 352
IBM Toolbox per Java 2 Micro Edition 353
Scaricamento ed impostazione di ToolboxME per iSeries 353
Concetti importanti per l'utilizzo di ToolboxME per iSeries 353
Classi ToolboxME per iSeries 355
Creazione ed esecuzione del programma ToolboxME per iSeries 366
Esempi di lavoro di ToolboxME per iSeries . . 380
Componenti XML (Extensible Markup Language) 381
PMCL (Program Call Markup Language) . . . 381
RFML (Record Format Markup Language) . . . 408
Programma di analisi XML e processore XSLT 417
XPCML (Extensible Program Call Markup Language) 418
FAQ (Frequently asked questions) 446
Suggerimenti per la programmazione 446
Chiusura del programma Java 447
Nomi percorso IFS per gli oggetti server . . . 447
Gestione dei collegamenti 449
JVM (Java Virtual Machine) OS/400 459
IASP (Independent auxiliary storage pool) . . . 464
Ottimizzazione di OS/400 464
Miglioramenti delle prestazioni 467
National language support di Java 469
Servizio e supporto per IBM Toolbox per Java 469
Esempi di codice 470
Esempi: classi Access 470
Esempi: JavaBean 539
Esempi: classi Commtrace 547
Esempi di Graphical Toolbox 547
Esempi dalle classi HTML 591
Esempi: PCML (Program Call Markup Language) 614
Esempi: classi ReportWriter 624
Esempi: classi Resource 640
Esempi: RFML 644
Esempio: utilizzo di una credenziale di token di profilo per scambiare l'identità del sottoprocesso OS/400 646
Esempi dalle classi servlet 646
Esempi semplici di programmazione 674
Esempi: suggerimenti per la programmazione 690
Esempi: ToolboxME per iSeries 691
Esempi: classi Utility 711
Esempi: classi Vaccess 716
Esempi: XPCML 746
Informazioni correlate per IBM Toolbox per Java 757
Informazioni sull'Esonero di responsabilità per gli esempi di codice 759
Disposizioni per il download e la stampa delle pubblicazioni 760

IBM Toolbox per Java

IBM Toolbox per Java è una serie di classi Java(TM) che consentono di utilizzare programmi Java per accedere ai dati sui server iSeries. E' possibile utilizzare tali classi per scrivere applicazioni client/server, applet e servlet che gestiscono i dati su iSeries. E' inoltre possibile eseguire le applicazioni Java che utilizzano le classi IBM Toolbox per Java sulla JVM di iSeries .

IBM Toolbox per Java utilizza i Server host di iSeries ../rzaii/rzaiihostserver.htm come punti di accesso al sistema. Poiché IBM Toolbox per Java utilizza funzioni di comunicazione incorporate in Java, non è necessario utilizzare IBM @server iSeries Access per Windows per utilizzare IBM Toolbox per Java. Ogni server si esegue in un lavoro separato sul server e ogni lavoro del server invia e riceve flussi di dati su un collegamento socket.

E' possibile ottenere ulteriori informazioni su IBM Toolbox per Java utilizzando la barra di navigazione principale i seguenti collegamenti:

Novità in V5R3
Leggere informazioni relative alle modifiche significative, sulla funzionalità potenziata ed altri argomenti o note.

Stampa dell'argomento IBM Toolbox per Java
Visualizzare o scaricare un PDF per l'argomento IBM Toolbox per Java. E' inoltre possibile scaricare l'argomento IBM Toolbox per Java in un file compresso.

Utilizzo del rilevatore classe
Tale rilevatore è utile per ricercare facilmente e velocemente le classi per nome e per descrizione, visualizzare le classi per pacchetto o esaminare un elenco in ordine alfabetico di tutte le classi IBM Toolbox per Java.

Installazione e gestione di IBM Toolbox per Java
Acquisire informazioni sulla gestione dell'installazione di IBM Toolbox per Java. Apprendere le istruzioni su come istallarlo su stazioni di lavoro e server. Utilizzare gli esempi semplici di programmazione per iniziare a utilizzare le classi IBM Toolbox per Java nelle applicazioni.

Classi IBM Toolbox per Java
Leggere le informazioni sulle varie classi presenti nei pacchetti IBM Toolbox per Java che consentono di gestire i dati server iSeries. Queste informazioni includono spiegazioni, codici di esempio e informazioni tecniche che facilitano la creazione di programmi IBM Toolbox per Java.

Utilizzo di Graphical Toolbox per creare i pannelli GUI
Utilizzare Graphical Toolbox per creare pannelli di interfaccia utente personalizzati in Java, che è possibile incorporare nelle applicazioni e nelle applet Java o nei moduli aggiuntivi iSeries Navigator.

JavaBeans
Leggere informazioni sulla creazione di JavaBeans utilizzando le classi pubbliche IBM Toolbox per Java, che sono create in base agli standard JavaBean di Javasoft. Esaminare gli esempi che illustrano come utilizzare i JavaBeans nei programmi.

JDBC
Acquisire informazioni sul supporto JDBC fornito con IBM Toolbox per Java. Utilizzando JDBC, i programmi possono emettere istruzioni SQL (structured query language) ed elaborare i risultati dai database sui server.

Supporto proxy
Acquisire informazioni su come utilizzare il supporto proxy di IBM Toolbox per Java, che include l'utilizzo del protocollo SSL per la codifica dei dati.

Sicurezza
Acquisire informazioni sull'utilizzo di Java Secure Socket Extension (JSSE) e IBM Toolbox per Java per fornire uno scambio sicuro di dati tra client e server un protocollo di applicazione su TCP/IP.

Proprietà di sistema
Acquisire informazioni sull'utilizzo delle proprietà di sistema per configurare vari aspetti di IBM Toolbox per Java, ad esempio, quando si definisce un server proxy o un livello di traccia. E' possibile utilizzare le proprietà di sistema per eseguire la configurazione appropriata del tempo di esecuzione senza ricompilare il codice.

IBM Toolbox per Java 2 Micro Edition
Utilizzare questo nuovo componente IBM Toolbox per Java per scrivere programmi Java per varie unità senza fili. Tramite l'utilizzo di ToolboxME per iSeries, le unità senza fili posso accedere direttamente alle risorse e ai dati server di iSeries.

Componenti XML
Acquisire informazioni su componenti XML differenti in IBM Toolbox per Java che consentono più facilmente ai programmi Java di accedere e utilizzare i dati e le funzioni sul server iSeries.

Javadoc relativi a IBM Toolbox per Java
Visualizzare le informazioni di riferimento di javadoc per le classi IBM Toolbox per Java.

FAQ (Frequently asked questions)
Reperire le risposte alle domande sull'ottimizzazione delle prestazioni di IBM Toolbox per Java, eseguendo la risoluzione dei problemi, utilizzando JDBC e altro ancora.

Ulteriori informazioni includono:

- Suggerimenti di programmazione per facilitare l'utilizzo di IBM Toolbox per Java
- Un elenco di esempi di programmazione di IBM Toolbox per Java
- Informazioni correlate, che includono collegamenti ad ulteriori informazioni su Java, sui servlet, su XML e altro.

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per informazioni legali importanti.

Novità in V5R3

IBM Toolbox per Java è disponibile nei seguenti formati:

- Il programma su licenza relativo a IBM Toolbox per Java, 5722-JC1, Versione 5 Release 3 (V5R3) si installa su V5R1 e successive versioni di OS/400. Da un client, IBM Toolbox per Java si ricollega a V5R1 e versioni successive di OS/400.
- OS/400 include anche le classi non grafiche di IBM Toolbox per Java ottimizzate per l'utilizzo durante l'esecuzione delle classi di IBM Toolbox per Java su una JVM (Java virtual machine). Così, ad esempio, se non è necessaria la funzionalità grafica del programma su licenza, è possibile ancora utilizzare facilmente IBM Toolbox per Java. Per maggiori informazioni, consultare File jar.
- IBM Toolbox per Java è disponibile anche in una versione sorgente aperto. E' possibile scaricare il codice ed ottenere ulteriori informazioni dal sito Web JTOpen  .

Nuovi pacchetti

Il pacchetto com.ibm.as400.commtrace fornisce una serie di classi che consentono ai programmi Java di gestire i dati di traccia delle comunicazioni per una descrizione linea LAN (Ethernet o token ring) specificata. E' possibile trasferire i dati di traccia grezzi precedentemente sottoposti a dump in un file di flusso o in un file di emissione, formattarli ed analizzare i dati (grezzi o formattati) per estrarre le informazioni desiderate. E' inoltre possibile eseguire una delle classi come programma di utilità della riga comandi autonomo.

Il pacchetto com.ibm.as400.util fornisce classi utility oltre al contenuto del pacchetto dei programmi di utilità.

Nuove classi

IBM Toolbox per Java V5R3 dispone di nuove classi nei pacchetti esistenti. Le nuove classi consentono di:

- Utilizzare la classe HTMLDocument con le classi HTML di IBM Toolbox per Java esistenti per creare pagine o documenti HTML che contengono tag FO (Formatting Object) XSL (Extensible Stylesheet Language). Utilizzando tag FO XSL è possibile trasformare il documento in altri tipi di documenti, come ad esempio un documento PDF (Portable Document Format).
- Utilizzare la classe IFSFileSystemView come un gateway grafico all'IFS (integrated file system) sul server. Utilizzare IFSFileSystemView quando si desidera creare oggetti javax.swing.JFileChooser, che è una modalità Java standard per creare finestre di dialogo per esaminare e scegliere file. IFSFileSystemView sostituisce IFSFileDialog. IBM Toolbox per Java supporta ancora IFSFileDialog ma consiglia invece di utilizzare IFSFileSystemView.
- Utilizzare la classe CommandHelpRetriever per creare documentazione di aiuto sui comandi CL formattati da IBM. E' possibile eseguire CommandHelpRetriever da una riga comandi o incorporare la funzione nel proprio programma Java.

Classi potenziate

IBM Toolbox per Java V5R3 inoltre include aggiornamenti alle classi esistenti. Gli aggiornamenti offrono:

- Aggiornamenti nelle classi HTML facilitano la creazione di dati sorgente HTML o FO XSL e la creazione di tag head per le pagine HTML.

Per informazioni sulle funzioni JDBC potenziate, consultare Aggiornamenti al supporto JDBC di IBM Toolbox per Java

Nuovo componente XML

In V5R3, IBM Toolbox per Java ha aggiunto l'XPCML (Extensible Program Call Markup Language). Supportando gli schemi XML, XPCML offre funzionalità e possibilità di utilizzo potenziate rispetto a PCML (Program Call Markup Language). Ad esempio, è possibile utilizzare XPCML per specificare e passare valori per i parametri di programma, richiamare i risultati di una chiamata al programma nel server iSeries in XPCML ed altro ancora.

Compatibilità

IBM Toolbox per Java non fornisce più x4j400.jar (IBM XML parser). Si consiglia l'utilizzo di uno dei seguenti programmi di analisi XML compatibili con JAXP da parte delle applicazioni:

- Il programma di analisi XML creato in JDK 1.4 e successivi
- Il programma di analisi XML Apache Xerces disponibile all'indirizzo xml.apache.org
- Uno dei programmi di analisi XML forniti con OS/400 in /QIBM/ProdData/OS400/xml/lib

IBM Toolbox per Java non supporta più l'esecuzione nella JVM predefinita in Netscape(R) Navigator o Microsoft(R) Internet Explorer. L'esecuzione di applet che utilizzano classi di IBM Toolbox per Java in un browser richiede di installare un modulo aggiuntivo, come ad esempio il più recente modulo aggiuntivo JRE (Java 2 Runtime Environment) Sun .

IBM Toolbox per Java non include più data400.jar. Le classi contenute in data400.jar ora si trovano in jt400.jar. Rimuovere data400.jar dalle istruzioni CLASSPATH.

I metodi getObject() per ResultSet e CallableStatement restituiscono ora oggetti Integer quando SQLType è SMALLINT. Nelle versioni precedenti questi metodi restituivano oggetti Short. Se si utilizza readObject per la lettura delle colonne SMALLINT, è necessario modificare l'applicazione Java per adattare il nuovo tipo di oggetto restituito.

Una diversa notifica dell'errore quando si immettono errori di troncamento dati genera messaggi di avvertenza che non causano l'errore dell'applicazione.

Non è possibile utilizzare questo release di IBM Toolbox per Java per deserializzare alcuni oggetti serializzati tramite i release precedenti a V5R1.

Se si sta utilizzando SSL (Secure Sockets Layer) per codificare i dati che circolano tra il client e il server, è necessario utilizzare uno dei seguenti componenti:

- JSSE (Java Secure Socket Extension)
- Oggetti SSL distribuiti nella V5R1 o in una versione successiva del programma su licenza IBM iSeries Client Encryption 5722-CE2 o 5722-CE3. Questo release di IBM Toolbox per Java non funziona con la V4R5 e con le versioni precedenti di iSeries Client Encryption.

Per utilizzare tutte le classi IBM Toolbox per Java, utilizzare J2SE (Java 2 Platform Standard Edition). L'utilizzo delle classi vaccess o di Graphical Toolbox richiede l'uso del pacchetto Swing, fornito con J2SE. L'utilizzo di PDML richiede che si esegua la versione 1.4 o successive di Java Runtime Environment.

Per ulteriori informazioni, esaminare i Requisiti OS/400 per l'esecuzione di IBM Toolbox per Java.

Come riconoscere novità e modifiche

Per individuare le modifiche apportate, queste informazioni (ma non i javadoc) utilizzano i seguenti simboli:

- » contrassegna l'inizio delle informazioni nuove o modificate.
- « contrassegna la fine delle informazioni nuove o modificate.

Per ulteriori informazioni sulle novità o le modifiche in questo release, consultare Memo per gli utenti.

Stampa di questo argomento

Per visualizzare o scaricare la versione PDF, selezionare PDF IBM Toolbox per Java (circa 2.8 megabyte).

Nota: l'argomento IBM Toolbox per Java contiene alcune informazioni che non sono incluse nei file PDF.

Salvataggio dei file PDF

Per salvare un PDF sulla stazione di lavoro per la visualizzazione o per la stampa:

- Fare clic con il tastino destro del mouse sul PDF nel proprio browser (fare clic con il tastino destro del mouse sul collegamento sopra riportato).
- Fare clic su **Salva destinazione con nome...** se si sta utilizzando Internet Explorer. Fare clic su **Salva collegamento con nome...** se si sta utilizzando Netscape Communicator.
- Passare all'indirizzario in cui si desidera salvare il PDF.
- Fare clic su **Salva**.

Scaricamento di Adobe Acrobat Reader

E' necessario disporre del programma Adobe Acrobat Reader per visualizzare o stampare questi PDF. E' possibile scaricarne una copia dal sito web di Adobe (www.adobe.com/products/acrobat/readstep.html) .

Scaricamento delle informazioni su IBM Toolbox per Java in un pacchetto soppresso

E' possibile scaricare un pacchetto soppresso dell'argomento IBM Toolbox per Java che comprende javadocs dal sito web di IBM Toolbox per Java e JTOpen  .

Nota: le informazioni nel pacchetto compresso dispongono di collegamenti a documenti non compresi nel pacchetto compresso, pertanto tali collegamenti non sono attivi.

Installazione e gestione di IBM Toolbox per Java

L'utilizzo di IBM Toolbox per Java facilita la scrittura di applet Java del client, servlet e applicazioni che hanno accesso a risorse, dati e programmi iSeries.

Le informazioni che seguono aiuteranno l'utente ad installare e ad iniziare ad utilizzare IBM Toolbox per Java:

Gestione dell'installazione
Verificare come differenti modalità di effettuare e gestire l'installazione di IBM Toolbox per Java influenzano la facilità di gestione e le prestazioni.

Installazione di IBM Toolbox per Java
Acquisire informazioni sui requisiti di OS/400 e della stazione di lavoro per l'installazione di IBM Toolbox per Java in un ambiente client/server. Acquisire conoscenze su come installare IBM Toolbox per Java nei server iSeries e nelle stazioni di lavoro.

Proprietà di sistema
Acquisire informazioni sulle proprietà di sistema e conoscenze su come utilizzarle per configurare vari aspetti di IBM Toolbox per Java.

Esempi semplici di programmazione
Introduzione all'utilizzo di IBM Toolbox per Java. Creare il primo programma IBM Toolbox per Java o esaminare gli esempi semplici di programmazione. Gli esempi mostrano come iniziare ad utilizzare IBM Toolbox per Java per gestire i dati ed i servizi disponibili sul server iSeries.

Gestione dell'installazione di IBM Toolbox per Java

E' necessario installare IBM Toolbox per Java solo su sistemi client che lo utilizzano o si trovano in un'ubicazione sulla rete a cui i client possono accedere. I client possono essere personal computer, stazioni di lavoro dedicate o sistemi iSeries. E' importante ricordarsi che è possibile configurare un server iSeries o una partizione del server in modo che funga da client. In quest'ultimo caso, è necessario installare IBM Toolbox per Java sulla partizione client del server.

Si può utilizzare uno qualsiasi dei seguenti metodi (a solo o in combinazione) per installare e gestire IBM Toolbox per Java:

- Gestione individuale per installare e gestire individualmente IBM Toolbox per Java su ogni client
- Gestione di rete di installazioni client utilizzando AS400ToolboxInstaller per installare e gestire IBM Toolbox per Java su ogni client
- Gestione di rete di una singola installazione utilizzando la rete per installare e gestire una singola installazione, condivisa di IBM Toolbox per Java su un server

Le sezioni che seguono spiegano brevemente come ogni metodo influenzi sia le prestazioni che la gestibilità. Il modo in cui si decide di sviluppare le applicazioni Java e gestire le risorse determina quale metodo (o quale combinazione di metodi) utilizzare.

Gestione individuale

E' possibile scegliere di gestire individualmente le istallazioni di IBM Toolbox per Java su singoli client. Il vantaggio principale di installare IBM Toolbox per Java su singoli client è che riduce il tempo che un client impiega ad avviare un'applicazione che utilizza classi IBM Toolbox per Java.

Lo svantaggio principale è costituito dalla gestione individuale di tali istallazioni. Un utente o un'applicazione che si crea deve tenere traccia della versione di IBM Toolbox per Java installata su ogni stazione di lavoro e gestirla.

Gestione di rete delle installazioni client

E' possibile scegliere di utilizzare la rete e AS400ToolboxInstaller per gestire le istallazioni client di IBM Toolbox per Java. Poiché ogni client dispone della propria copia di IBM Toolbox per Java, questo tipo di installazione presenta lo stesso vantaggio di riduzione del tempo impiegato da un client per avviare un'applicazione IBM Toolbox per Java. Rende anche possibile aggiornare automaticamente tutte le installazioni individuali di IBM Toolbox per Java.

Lo svantaggio principale è costituito dal creare ed effettuare la manutenzione dell'elaborazione che utilizza AS400ToolboxInstaller per gestire le installazioni individuali.

Gestione di rete di una singola installazione

E' anche possibile utilizzare la propria rete per installare e gestire una singola copia di IBM Toolbox per Java su un server a cui tutti i client possono accedere. Questo tipo di installazione di rete fornisce i seguenti vantaggi:

- Tutti i client utilizzano la stessa versione di IBM Toolbox per Java
- L'aggiornamento di una singola installazione di IBM Toolbox per Java è vantaggioso per tutti i client
- I client individuali non hanno problemi di manutenzione, a parte l'impostazione dello stesso CLASSPATH iniziale

Questo tipo di installazione presenta anche lo svantaggio di aumentare il tempo che un client impiega per avviare un'applicazione IBM Toolbox per Java. E' necessario inoltre abilitare il CLASSPATH del client in modo che punti al server. E' possibile utilizzare iSeries NetServer, integrato in OS/400 o un metodo differente che consente all'utente di accedere ai file sui server iSeries, come ad esempio iSeries Access per Windows.

Installazione di IBM Toolbox per Java

L'installazione di IBM Toolbox per Java dipende da come si desidera gestire l'installazione.Dopo aver deciso come gestire l'installazione, assicurarsi che l'ambiente soddisfi i requisiti che seguono:

- Requisiti OS/400
- Requisiti della stazione di lavoro

Installazione di IBM Toolbox per Java

Una volta deciso come si desidera gestire l'installazione e considerati i requisiti per l'esecuzione di IBM Toolbox per Java, è possibile installare IBM Toolbox per Java:

- Installazione di IBM Toolbox per Java sul server
- Installazione di IBM Toolbox per Java sulla stazione di lavoro

Requisiti OS/400 per IBM Toolbox per Java

Dopo aver deciso come gestire l'installazione, assicurarsi che l'ambiente OS/400 soddisfi i requisiti che seguono:

- Opzioni OS/400 necessarie
- Dipendenze su altri programmi su licenza
- Compatibilità con livelli diversi di OS/400
- Ottimizzazioni native durante l'esecuzione su JVM OS/400

- Requisiti per l'esecuzione di applicazioni ToolboxME per iSeries

Nota: prima di utilizzare IBM Toolbox per Java, assicurarsi di considerare i requisiti stazione di lavoro appropriati al proprio ambiente.

Opzioni OS/400 necessarie:

L'esecuzione di IBM Toolbox per Java in un ambiente client/server richiede l'abilitazione del profilo utente QUSER, l'avvio dei server host e TCP/IP in esecuzione:

- Il profilo utente QUSER deve essere abilitato per avviare i server host.
- I server host ricevono e accettano le richieste di collegamento dai client. L'opzione Server host OS/400 (prodotto su licenza 5722SS1) è inclusa con l'opzione di base di OS/400. Per ulteriori informazioni, consultare Gestione del server host.
- Il supporto TCP/IP, integrato ad OS/400, consente di collegare il server alla rete. Per ulteriori informazioni, consultare TCP/IP.

Avvio delle opzioni OS/400 richieste

Da una riga comandi iSeries, avviare le opzioni OS/400 necessarie completando le fasi che seguono:

1. Assicurarsi che il profilo QUSER sia abilitato.
2. Per avviare i server host OS/400, utilizzare il comando CL Avvio server host. Immettere **STRHOSTSVR *ALL** e premere **INVIO**.
3. Per avviare il server DDM (distributed data management) TCP/IP, utilizzare il comando CL Avvio server TCP/IP. Immettere **STRTCPSVR SERVER(*DDM)** e premere **INVIO**.

Come determinare se IBM Toolbox per Java è installato sul server:

Molti server iSeries vengono consegnati con il prodotto su licenza IBM Toolbox per Java già installato.

Per controllare se IBM Toolbox per Java è già installato, completare i seguenti passi:

1. In iSeries Navigator, selezionare il sistema che si desidera utilizzare e collegarsi ad esso.
2. Sull'**Albero delle funzioni** (il pannello di sinistra), espandere il sistema, quindi espandere **Configurazione e Servizio**.
3. Espandere **Software**, quindi espandere **Prodotti installati**.
4. Nel pannello **Dettagli** (il pannello di destra), cercare nella colonna **Prodotto** 5722jc1.Se si individua questo prodotto, il programma su licenza IBM Toolbox per Java è installato sul server selezionato.

   Nota: è anche possibile verificare se IBM Toolbox per Java è installato utilizzando il comando CL **GO MENU(LICPGM)** (Go to Menu), Opzione 11.

Se IBM Toolbox per Java non è installato, è possibile installare il prodotto su licenza IBM Toolbox per Java.

Se è installata una versione precedente di IBM Toolbox per Java, cancellare prima la versione attualmente installata, quindi installare il prodotto su licenza IBM Toolbox per Java. Per evitare possibili problemi, considerare la possibilità di effettuare una copia di riserva della versione attualmente installata di IBM Toolbox per Java prima di cancellarla.

Verifica del profilo QUSER:

I Server host OS/400 vengono avviati sotto il profilo utente QUSER, quindi sarà necessario in primo luogo accertarsi che il profilo QUSER sia abilitato.

Verifica del profilo QUSER

Per utilizzare la riga comandi per verificare il profilo QUSER, completare le seguenti operazioni:

1. Su una riga comandi iSeries, immettere `DSPUSRPRF USRPRF(QUSER)` e premere **Invio**.
2. Verificare che lo **Stato** sia *ENABLED.Se lo stato del profilo non è *ENABLED, modificare il profilo QUSER.

Modifica del profilo utente QUSER:

Se il profilo QUSER non è *ENABLED, è necessario abilitarlo per avviare i server dell'host OS/400. Inoltre, la parola d'ordine del profilo QUSER non può essere *NONE. In questo caso, è necessario reimpostarla.

Per utilizzare la riga comandi per abilitare il profilo QUSER, completare le seguenti operazioni:

1. Immettere `CHGUSRPRF USRPRF(QUSER)` e premere **INVIO**.
2. Modificare il campo **Stato** in *ENABLED e premere **Invio**.

Il profilo utente QUSER è ora pronto per avviare i server dell'host OS/400.

Dipendenze su altri programmi su licenza:

A seconda di come si desidera utilizzare IBM Toolbox per Java, potrebbe essere necessario installare altri programmi su licenza. Le informazioni che seguono descrivono queste dipendenze.

Programma di visualizzazione file di spool

Quando si desidera utilizzare le funzioni del programma di visualizzazione file di spool (classe SpooledFileViewer) di IBM Toolbox per Java, assicurarsi che sia stata installata l'opzione host 8 (Font di compatibilità AFP) sul server.

Nota: le classi SpooledFileViewer, PrintObjectPageInputStream e PrintObjectTransformedInputStream funzionano solo quando sono collegate a V4R4 o sistemi successivi.

SSL (Secure Sockets Layer)

Quando si desidera utilizzare SSL (Secure Sockets Layer), assicurarsi di aver installato quanto segue:

- IBM HTTP Server per il programma su licenza iSeries, 5722-DG1
- Opzione OS/400 34 (Digital Certificate Manager)
- IBM Cryptographic Access Provider 128-bit per iSeries, 5722-AC3
- iSeries Client Encryption (128-bit), 5722-CE3

La versione V5R3 di IBM Toolbox per Java richiede che si utilizzi la versione V5R1, V5R2 o V5R3 di iSeries Client Encryption.

Per ulteriori informazioni su SSL, consultare "SSL (Secure Sockets Layer) e JSSE (Java Secure Socket Extension)" a pagina 352.

Server HTTP per l'utilizzo di applet, servlet, SSL o AS400ToolboxInstaller

Se si desidera utilizzare applet, servlet, SSL o la classe AS400ToolboxInstaller sul server iSeries, è necessario impostare un server HTTP ed installare i file di classe sul server iSeries. Per ulteriori informazioni su IBM HTTP Server, consultare IBM HTTP Server for AS/400 Webmaster's Guide, GC41-5434, al seguente URL: http://www.ibm.com/eserver/iseries/products/http/docs/doc.htm


.La Webmaster's Guide è disponibile sia in formato HTML che PDF.

Per informazioni su Digital Certificate Manager e su come creare e gestire certificati digitali utilizzando IBM HTTP Server, consultare Gestione dei certificati digitali.

Compatibilità con livelli diversi di OS/400:

Dal momento che è possibile utilizzare IBM Toolbox per Java sia sul server che sul client, i problemi di compatibilità influenzano sia l'esecuzione su un server che il collegamento inverso da un client ad un server.

Esecuzione di IBM Toolbox per Java, Versione 5 Release 3, sui server

Per installare IBM Toolbox per Java (programma su licenza 5722-JC1 V5R3M0) su un sistema iSeries, sul server deve essere in esecuzione uno dei seguenti sistemi:

- OS/400 Versione 5 Release 3
- OS/400 Versione 5 Release 2
- OS/400 Versione 5 Release 1

E' possibile installare sul sistema solo una versione del programma su licenza IBM Toolbox per Java. Per installare una versione diversa, è necessario prima eliminare il programma su licenza IBM Toolbox per Java già esistente.

Utilizzo di IBM Toolbox per Java per effettuare il collegamento inverso da un client ad un server

E' possibile utilizzare versioni differenti di IBM Toolbox per Java su un client e sul server a cui ci si sta collegando. Per utilizzare Versione 5 Release 3 di IBM Toolbox per Java per accedere a dati e risorse su un sistema iSeries, sul **server a cui ci si sta collegando** deve essere in esecuzione uno dei seguenti sistemi:

- OS/400 Versione 5 Release 3
- OS/400 Versione 5 Release 2
- OS/400 Versione 5 Release 1

La tabella che segue mostra i requisiti di compatibilità per l'installazione di IBM Toolbox per Java e per il collegamento inverso a versioni differenti di OS/400.

Nota: IBM Toolbox per Java non supporta la compatibilità con versioni successive. Non è possibile installare IBM Toolbox per Java su un server su cui è in esecuzione una versione più recente di OS/400 né utilizzarlo per collegarsi ad esso. Ad esempio, se si sta utilizzando la versione di IBM Toolbox per Java che si invia con OS/400 V5R1, non è possibile installarla su un un server su cui è in esecuzione OS/400 V5R3 o collegarla ad esso.

LPP	Viene fornito con OS/400	Si installa su OS/400	Ritorna a collegarsi ad OS/400
5722-JC1 V5R1M0	V5R1	V4R4 e successive	V4R3 e successive
5722-JC1 V5R2M0	V5R2	V4R5 e successive	V4R5 e successive
5722-JC1 V5R3M0	V5R3	V5R1 e successive	V5R1 e successive

Ottimizzazioni native durante l'esecuzione su JVM iSeries:

Le ottimizzazioni native sono una serie di funzioni che permettono alle classi IBM Toolbox per Java di funzionare nel modo previsto quando si eseguono su OS/400. Le ottimizzazioni influenzano l'operatività di IBM Toolbox per Java solo quando vengono eseguite su JVM iSeries.

E' molto importante capire che i programmi Java utilizzano ottimizzazioni native solo quando si usa una versione di IBM Toolbox per Java corrispondente alla versione di OS/400 sul server. Le ottimizzazioni sono:

- Collegamento: quando non sono specificati l'ID utente o la parola d'ordine nell'oggetto AS400, vengono utilizzati l'ID utente e la parola d'ordine del lavoro corrente
- Chiamata diretta alle API OS/400 invece di effettuare chiamate socket ai server host:
  - Accesso al database a livello record, code di dati e spazio utente quando vengono soddisfatti i requisiti di sicurezza.
  - Chiamata al programma e chiamata al comando quando i requisiti di sicurezza e i requisiti di protezione durante il sottoprocesso vengono soddisfatti.

Nota: per ottenere le migliori prestazioni, impostare le proprietà dell'unità di controllo JDBC in modo da utilizzare l'unità di controllo nativa quando il programma Java e il file di database si trovano sullo stesso server iSeries.

Non è necessaria alcuna modifica all'applicazione Java per richiamare le ottimizzazioni. IBM Toolbox per Java abilita automaticamente le ottimizzazioni quando necessario.

Requisiti di compatibilità dell'ottimizzazione nativa

La seguente tabella indica quali versioni di LPP IBM Toolbox per Java ed OS/400 si devono eseguire per utilizzare le ottimizzazioni native. Questa tabella documenta solo le questioni di compatibilità che influenzano le ottimizzazioni native. Per questioni di compatibilità generale, consultare Compatibilità con livelli diversi di OS/400.

Livello di OS/400	**LPP IBM Toolbox per Java richiedeva l'utilizzo di ottimizzazioni native**		
V5R1	5722-JC1 V5R1M0		
V5R2		5722-JC1 V5R2M0	
V5R3			5722-JC1 V5R3M0

Per ottenere miglioramenti delle prestazioni, è necessario assicurarsi di utilizzare il file jar che include le ottimizzazioni native OS/400. Per ulteriori informazioni, consultare la Nota 1 in File jar.

Quando le versioni di IBM Toolbox per Java e OS/400 non corrispondono, le ottimizzazioni native non sono disponibili. In questo caso, IBM Toolbox per Java funziona come se fosse eseguito su un client.

Requisiti ToolboxME per iSeries:

La stazione di lavoro, l'unità senza fili ed il server devono rispondere a determinati requisiti (elencati di seguito) per lo sviluppo e l'esecuzione delle applicazioni ToolboxME per iSeries. Anche se IBM Toolbox per Java 2 Micro Edition è considerato parte di IBM Toolbox per Java, esso non è incluso nel prodotto su licenza.

ToolboxME per iSeries (jt400Micro.jar) è incluso in una versione sorgente aperta di Toolbox per Java, denominata JTOpen. E' necessario scaricare ed installare separatamente ToolboxME per iSeries, contenuto in JTOpen.

Requisiti

Per poter utilizzare ToolboxME per iSeries, la stazione di lavoro, l'unità senza fili Tier0 ed il server devono soddisfare i seguenti requisiti.

Requisiti della stazione di lavoro

Requisiti della stazione di lavoro per sviluppare le applicazioni di ToolboxME per iSeries:
- Java 2 Platform, Standard Edition, versione 1.3 o successive
- JVM (Java Virtual Machine) per le unità senza fili
- Simulazione ed emulazione dell'unità senza fili

Requisiti dell'unità senza fili

Il solo requisito per l'esecuzione delle applicazioni di ToolboxME per iSeries sull'unità Tier0 è l'utilizzo di una JVM per le unità senza fili.

Requisiti del server

Requisiti del server per utilizzare le applicazioni di ToolboxME per iSeries:
- Classe MEServer, inclusa in IBM Toolbox per Java o l'ultima versione di JTOpen
- Requisiti OS/400 per IBM Toolbox per Java

Requisiti della stazione di lavoro per IBM Toolbox per Java

Dopo aver deciso come gestire l'installazione, assicurarsi che la stazione di lavoro soddisfi i requisiti che seguono:
- Requisiti per l'esecuzione delle applicazioni Java
- Requisiti per l'esecuzione delle applet Java
- Requisiti per lo sviluppo di ToolboxME per le applicazioni iSeries
- Requisiti Swing

Nota: prima di utilizzare IBM Toolbox per Java, accertarsi di considerare i requisiti OS/400 appropriati al proprio ambiente.

Requisiti stazione di lavoro per l'esecuzione di applicazioni IBM Toolbox per Java:

Per sviluppare ed eseguire applicazioni IBM Toolbox per Java, accertarsi che la stazione di lavoro soddisfi i seguenti requisiti:
- Si consiglia l'utilizzo di un JVM (Java virtual machine) J2SE (Java 2 Standard Edition). Molte nuove funzioni IBM Toolbox per Java richiedono l'utilizzo della versione 1.4 o superiori della JVM.
- L'utilizzo delle classi vaccess o di Graphical Toolbox richiede Swing, che viene fornito con J2SE. E' possibile anche scaricare Swing 1.1 dal sito Web Classi Foundation Java Sun . Gli ambienti che seguono sono stati esaminati:
  - Windows(R) 2000
  - Windows(R) XP
  - AIX Versione 4.3.3.1
  - Sun Solaris(TM) Versione 5.7
  - OS/400 Versione 5 Release 1 o successivi
  - Linux (Red Hat 7.0)
- TCP/IP installato e configurato

Requisiti della stazione di lavoro per eseguire le applet IBM Toolbox per Java:

Per sviluppare ed eseguire applicazioni IBM Toolbox per Java, accertarsi che la stazione di lavoro soddisfi i seguenti requisiti:
- Un browser che dispone di una JVM (Java virtual machine) compatibile. Gli ambienti che seguono sono stati esaminati:

– Netscape Communicator 4.7, che utilizza il modulo aggiuntivo Java 1.3 o successivi

Nota: IBM Toolbox per Java non supporta più l'esecuzione nella JVM predefinita, in Netscape Navigator o Microsoft Internet Explorer. Per l'applet che utilizza classi di IBM Toolbox per Java per l'esecuzione in un browser, si dovrebbe installare un modulo aggiuntivo come ad esempio il modulo aggiuntivo Sun JRE (Java 2 Runtime Environment) .

- TCP/IP installato e configurato
- La stazione di lavoro deve collegarsi ad un server che esegue OS/400 V5R1 o successive versioni

Requisiti Swing della stazione di lavoro per IBM Toolbox per Java:

IBM Toolbox per Java è passato a supportare Swing 1.1 in V4R5 e questo release continua il supporto in questione. Il passaggio a Swing ha richiesto modifiche di programmazione nelle classi IBM Toolbox per Java. Quindi, se i programmi utilizzano il Graphical Toolbox o le classi vaccess da release precedenti a V4R5, è necessario modificare anche i programmi.

In aggiunta ad una modifica di programmazione, le classi Swing devono trovarsi in CLASSPATH quando il programma viene eseguito. Le classi Swing sono parte di Java 2 Platform. Se non si dispone di Java 2 Platform, è possibile scaricare le classi Swing 1.1 da Sun Microsystems, Inc.

Installazione di IBM Toolbox per Java su un server iSeries

E' necessario installare IBM Toolbox per Java sul server iSeries solo quando è stato configurato il server o una sua partizione come client.

Nota: la versione nativa di IBM Toolbox per Java viene fornita con OS/400. Quindi, se si desidera utilizzare IBM Toolbox per Java solo sul server iSeries, non è necessario installare il prodotto su licenza. Per ulteriori informazioni sulla versione nativa di IBM Toolbox per Java, consultare File Jar: Nota 1.

Prima di installare IBM Toolbox per Java, è necessario accertarsi che la versione di OS/400 soddisfi i requisiti per l'esecuzione di IBM Toolbox per Java. Inoltre, alcuni server vengono forniti preconfigurati con un'installazione di IBM Toolbox per Java. Si potrebbe voler determinare se il prodotto su licenza IBM Toolbox per Java è già installato sul server.

Installazione di IBM Toolbox per Java

E' possibile installare il programma su licenza Toolbox per Java utilizzando iSeries Navigator o la riga comandi.

Utilizzo di iSeries Navigator per l'installazione di IBM Toolbox per Java

Per installare IBM Toolbox per Java utilizzando iSeries Navigator, completare i seguenti passi:

1. In iSeries Navigator, collegarsi al sistema che si desidera utilizzare.
2. Sull'Albero delle funzioni (pannello a sinistra), espandere **Collegamenti**.
3. In **Collegamenti**, fare clic col tastino destro sul sistema su cui si desidera installare IBM Toolbox per Java.
4. Selezionare **Esegui comando**.
5. Nella finestra di dialogo **Ripristino del programma su licenza (RSTLICPGM)**, immettere le informazioni che seguono, poi fare clic su **OK**:
   - Prodotto: 5722JC1
   - Unità: il nome dell'unità o il file di salvataggio

Nota: per ulteriori informazioni, fare clic su **Aiuto** nella finestra di dialogo **Ripristino del programma su licenza (RSTLICPGM)**.

E' possibile utilizzare iSeries Navigator per visualizzare lo stato dell'attività del comando Management Central risultante completando le fasi che seguono:

1. Espandere **Management Central**.
2. Espandere **Funzione attività**.
3. In **Funzione attività**, selezionare **Comandi**.
4. Nel pannello Dettagli, fare clic sull'attività **Esegui comando** appropriata.

Utilizzo della riga comandi per l'installazione di IBM Toolbox per Java

Per installare IBM Toolbox per Java da una riga comandi iSeries, completare i seguenti passi:

1. Su una riga comandi iSeries, utilizzare il comando CL Go to Menu. Immettere **GO MENU(LICPGM)** e premere **INVIO**.
2. Selezionare **11.Installazione programma su licenza**.
3. Selezionare **5722-JC1 IBM Toolbox per Java**.

Per ulteriori informazioni sull'installazione di programmi su licenza, consultare Gestione del software e dei programmi su licenza.

Installazione di IBM Toolbox per Java sulla stazione di lavoro

Prima di installare IBM Toolbox per Java, assicurarsi di considerare i requisiti della stazione di lavoro che si riferiscono all'ambiente.La modalità di installazione di IBM Toolbox per Java sulla stazione di lavoro dipende da come si desidera gestire l'installazione:

- Per installare IBM Toolbox per Java su singoli client, copiare i file JAR nella stazione di lavoro e configurare CLASSPATH della stazione di lavoro.
- Per utilizzare IBM Toolbox per Java che è installato su un server, si deve solo configurare CLASSPATH della stazione di lavoro in modo che punti all'installazione server. Per puntare al CLASSPATH della stazione di lavoro sul server, il server deve avere iSeries Netserver installato.

Questa documentazione spiega come copiare i file classe sulla stazione di lavoro.Per ulteriori informazioni sull'impostazione di CLASSPATH sulla stazione di lavoro, fare riferimento alla documentazione del sistema operativo per la stazione di lavoro o alle informazioni disponibili sul sito web Sun Java  .

Nota: l'utilizzo delle classi IBM Toolbox per Java nell'applicazione richiede anche che il sistema soddisfi i requisiti per OS/400.

I file di classe IBM Toolbox per Java vengono compressi in diversi file jar, di conseguenza è necessario copiare uno o più di questi file jar nella stazione di lavoro. Per ulteriori informazioni su quali file jar siano necessari per specifiche funzioni di IBM Toolbox per Java, consultare File jar.

Esempio: copia di jt400.jar

L'esempio che segue presuppone che l'utente voglia copiare jt400.jar, che contiene le classi fondamentali IBM Toolbox per Java.

Per copiare manualmente il file jar, completare le fasi che seguono:

1. Trovare il file jt400.jar nel seguente indirizzario: /QIBM/ProdData/HTTP/Public/jt400/lib
2. Copiare jt400.jar dal server sulla stazione di lavoro. E' possibile eseguire questa operazione in vari modi:

- Utilizzare iSeries Access per Windows per mettere in corrispondenza una unità di rete nella stazione di lavoro con il server, quindi copiare il file.
- Utilizzare l'FTP (File Transfer Protocol) per inviare il file (con il metodo binario) sulla stazione di lavoro.

3. Aggiornare la variabile di ambiente CLASSPATH della stazione di lavoro.
   - Ad esempio, se si sta utilizzando Windows NT e si è copiato jt400.jar su C:\jt400\lib, aggiungere le stringhe che seguono alla fine di CLASSPATH:

     ```
     ;C:\jt400\lib\jt400.jar
     ```

L'utente dispone inoltre dell'opzione per utilizzare la versione origine aperta di IBM Toolbox per Java, denominata JTOpen. Per ulteriori informazioni relative a JTOpen, consultare il sito web IBM Toolbox per Java and JTOpen  .

File Jar:

IBM Toolbox per Java viene fornito come una serie di file jar. Ogni file jar contiene pacchetti Java che forniscono funzioni specifiche. E' possibile ridurre la quantità di spazio di memorizzazione necessario utilizzando solo i file jar richiesti per abilitare le funzioni specifiche.

Per utilizzare un file jar, assicurarsi di inserire una voce ad esso relativa nel CLASSPATH.

La seguente tabella, indica quali file jar è necessario aggiungere al CLASSPATH per utilizzare la funzione associata o il pacchetto.

Pacchetto o funzioni di IBM Toolbox per Java	**File jar necessari in CLASSPATH**
Classi Access	jt400.jar (client) o jt400Native.jar (server) Nota 1 o jt400Proxy.jar in un ambiente proxy
"Classe CommandHelpRetriever" a pagina 252	jt400.jar (client) o jt400Native.jar (server) Nota 1 e un programma di analisi XML e un processore XSLT Nota 2
CommandPrompter Nota 3	jt400.jar, jui400.jar, util400.jar Nota 4 e un programma di analisi XML Nota 2
Classi Commtrace	jt400.jar (client) o jt400Native.jar (server) Nota 1
Classi HTML	jt400.jar Nota 1 più jt400Servlet.jar (client) o jt400Native.jar (server) Nota 1
Classe HTMLDocument	Gli stessi file .jar necessari per le classi HTML, più un programma di analisi XML e un processore XSLT Nota 2
Classi JCA	jt400.jar (client) o jt400Native.jar (server) Nota 1
GUI sorgente dati JDBC	jt400.jar (client) Nota 1 e jui400.jarNota 5
Sistema NLS e messaggi di errore	jt400Mri_lang_cntry.jar Nota 6
PCML (sviluppo e tempo di esecuzione, analizzato) Nota 7	jt400.jar (client) o jt400Native.jar (server) Nota 1, Nota 8 e un programma di analisi XML Nota 2
PCML (tempo di esecuzione, serializzato)	jt400.jar (client) o jt400Native.jar (server) Nota 1, Nota 8
PDML (sviluppo) Nota 3	uitools.jar, jui400.jar, util400.jar Nota 4 e un programma di analisi XML Nota 2
PDML (tempo di esecuzione, analizzato) Nota 3	jui400.jar, util400.jar Nota 4 e un programma di analisi XML Nota 2
PDML (tempo di esecuzione, serializzato) Nota 3	jui400.jar e util400.jar Nota 4
Classi ReportWriter	jt400.jar (client) o jt400Native.jar (server) Nota 1, file jar reportwriter Nota 9 e un programma di analisi XML e un processore XSLT Nota 2

Pacchetto o funzioni di IBM Toolbox per Java	File jar necessari in CLASSPATH
Classi Resource	jt400.jar (client) o jt400Native.jar (server) Nota 1
RFML	jt400.jar (client) o jt400Native.jar (server) Nota 1 e un programma di analisi XML Nota 2
Classi Security	jt400.jar (client) o jt400Native.jar (server) Nota 1 o jt400Proxy.jar in un ambiente proxy
Classi Servlet	jt400.jar Nota 1 più jt400Servlet.jar (client) o jt400Native.jar (server) Nota 1
iSeries System Debugger Nota 3	jt400.jar (client) Nota 1 e tes.jar
ToolboxME per iSeries	jt400Micro.jar (client) Nota 10 e jt400.jar (server) o jt400Native.jar (server)Nota 1
Classi Vaccess	jt400.jar (client) Nota 1
XPCML	jt400.jar (client) o jt400Native.jar (server) Nota 1 e un programma di analisi XML e un processore XSLT Nota 2

Nota 1: alcune classi IBM Toolbox per Java si trovano in più di un file jar:

- **jt400.jar** - Access, commtrace, JCA, supporto JDBC, MEServer, PCML, resource, RFML, security, utilities, vaccess e XPCML.
- **jt400.zip** - utilizzare jt400.jar invece di jt400.zip.jt400.zip viene fornito per mantenere la compatibilità con i release precedenti di IBM Toolbox per Java.
- **jt400Access.zip** - Le stesse classi che si trovano in jt400.jar ad eccezione delle classi vaccess. jtAccess400.zip viene fornito per mantenere la compatibilità con i release precedenti di IBM Toolbox per Java. Utilizzare jt400.jar o jt400Native.jar invece di jt400Access.zip.
- **jt400Native.jar** - Access, HTML, MEServer, PCML, resource, RFML, security, XPCML e ottimizzazioni origine. Le ottimizzazioni origine sono una serie di classi (meno di 20) che traggono profitto dalla funzione iSeries quando viene eseguita sulla JVM iSeries. Dal momento che jt400Native.jar contiene le ottimizzazioni origine, quando viene eseguito sulla JVM iSeries, utilizza jt400Native.jar invece di jt400.jar. jt400Native.jar viene fornito con OS/400 e risiede nell'indirizzario /QIBM/ProdData/OS400/jt400/lib.
- **jt400Native11x.jar** - solo Ottimizzazioni origine.Se si sta eseguendo la JVM iSeries e si desidera utilizzare jt400.jar, includere jt400Native11x.jar in CLASSPATH invece di jt400Native.jar. jt400Native11x.jar viene fornita con OS/400 e risiede in un indirizzario /QIBM/ProdData/OS400/jt400/lib.

Nota 2: quando è necessario utilizzare un programma di analisi XML o un processore XSLT, assicurarsi che siano compatibili con JAXP. Per ulteriori informazioni, consultare la seguente pagina:

"Programma di analisi XML e processore XSLT" a pagina 417

Nota 3: l'utilizzo di CommandPrompter, PDML o dell'iSeries System Debugger richiede un ulteriore file jar che non è parte dell'IBM Toolbox per Java: jhall.jar. Per ulteriori informazioni su come scaricare il file jhall.jar, consultare il sito Web Sun JavaHelp(TM) .

Nota 4: util400.jar contiene classi iSeries specifiche per formattare l'immissione e per utilizzare il programma di richiesta CL. L'utilizzo della classe CommandPrompter richiede util400.jar. Invece, per utilizzare PDML non è necessario util400.jar, comunque lo si consiglia.

Nota 5: jui400.jar contiene le classi necessarie per utilizzare l'interfaccia GUI DataSource JDBC.jt400.jar (Nota 1) contiene le classi necessarie per tutte le altre funzioni JDBC.

Nota 6: jt400Mri_xx_yy.jar contiene messaggi convertiti, incluse stringhe contenute in messaggi di eccezione, finestre di dialogo ed emissioni da altre elaborazioni normali. In jt400Mri_lang_cntry.jar, lang = il Codice lingua ISO e cntry = il Codice paese o regione ISO utilizzato per convertire il testo contenuto. In alcuni casi, il codice paese o regione ISO non viene utilizzato. La configurazione della versione di una NLV (National Language Version) particolare del programma su licenza IBM Toolbox per Java su iSeries prevede l'installazione del file jt400Mri_lang_cntry.jar appropriato. Se la lingua non è supportata, viene installata la versione predefinita in inglese, inclusa nei file jar IBM Toolbox per Java.

- Ad esempio, l'istallazione della versione in lingua tedesca del programma su licenza 5722-JC1 prevede il file jar in lingua tedesca, jt400Mri_de.jar.

E' possibile aggiungere un supporto per altre lingue aggiungendo più jar al percorso classe. Java carica la stringa esatta in base alla locale corrente.

Nota 7: la serializzazione del file PCML durante lo sviluppo ha due vantaggi:

1. E' necessario analizzare il file PCML solo durante lo sviluppo e non durante il tempo di esecuzione
2. Gli utenti necessitano di un numero inferiore di file jar in CLASSPATH per eseguire l'applicazione

Per analizzare il file PCML durante lo sviluppo, sono necessari sia il tempo di esecuzione PCML in data.jar o jt400.jar sia il programma di analisi PCML in x4j400.jar. Per eseguire l'applicazione serializzata, gli utenti necessitano solo di jt400.jar. Per ulteriori informazioni, consultare Creare chiamate di programma iSeries con PCML.

Nota 8: utilizzare jt400.jar e jt400Native.jar invece di data400.jar. data400.jar contiene le classi tempo di esecuzione PCML, che ora si trovano anche in jt400.jar e jt400Native.jar (Nota 1). data400.jar viene fornito per mantenere la compatibilità con i release precedenti di IBM Toolbox per Java.

Nota 9: le copie delle classi ReportWriter si trovano in più di un file jar:

- composer.jar
- outputwriter.jar
- reportwriters.jar

Se l'applicazione inserisce un flusso di dati PCL su un file di spool iSeries, è necessario rendere disponibili le classi Access utilizzando il file jar appropriato (Nota 1). La creazione di un file di spool per conservare i dati PCL richiede le classi AS400, OutputQueue, PrintParameterList e SpooledFileOutputStream. Per ulteriori informazioni, consultare Classi ReportWriter.

Nota 10: jt400Micro.jar non contiene le classi necessarie per eseguire MEServer, il quale risiede sia in jt400.jar che in jt400Native.jar (Nota 1). jt400Micro.jar è disponibile solo sul sito web IBM Toolbox per Java e JTOpen  .

Proprietà di sistema

E' possibile specificare le proprietà di sistema per configurare vari aspetti di IBM Toolbox per Java. Ad esempio, è possibile utilizzare le proprietà di sistema per definire un server proxy o un livello di traccia. Le proprietà di sistema sono utili per la configurazione del tempo di esecuzione senza la necessità di compilare nuovamente il codice. Le proprietà di sistema funzionano come le variabili d'ambiente, nel senso che quando si modifica una proprietà di sistema durante il tempo di esecuzione, tale modifica non si riflette prima della successiva esecuzione dell'applicazione.

Esistono diversi metodi per impostare le proprietà di sistema:

- **Utilizzare il metodo java.lang.System.setProperties()**

  E' possibile impostare le proprietà di sistema in modo programmatico utilizzando il metodo java.lang.System.setProperties().

Ad esempio, il seguente codice imposta la proprietà com.ibm.as400.access.AS400.proxyServer su hqoffice:

```
Properties systemProperties = System.getProperties();
   systemProperties.put ("com.ibm.as400.access.AS400.proxyServer", "hqoffice");
      System.setProperties (systemProperties);
```

- **Utilizzare l'opzione -D del comando java**

  Molti ambienti consentono di impostare le proprietà di sistema durante l'esecuzione delle applicazioni da una riga comandi utilizzando l'opzione -D del comando java.

  Ad esempio, il seguente programma esegue l'applicazione definita Inventario con la proprietà com.ibm.as400.access.AS400.proxyServer impostata su hqoffice:

  ```
  java -Dcom.ibm.as400.access.AS400.proxyServer=hqoffice Inventory
  ```

- **Utilizzare un file jt400.properties**

  In alcuni ambienti, potrebbe non essere conveniente consentire a tutti gli utenti di impostare le proprietà di sistema autonomamente. In alternativa, è possibile specificare le proprietà di sistema IBM Toolbox per Java in un file denominato jt400.properties che viene ricercato come se fosse parte del pacchetto com.ibm.as400.access. In altri termini, posizionare il file jt400.properties in un indirizzario com/ibm/as400/access indicato dal percorso classe.

  Ad esempio, impostare la proprietà com.ibm.as400.access.AS400.proxyServer su hqoffice inserendo la seguente riga nel file jt400.properties:

  ```
  com.ibm.as400.access.AS400.proxyServer=hqoffice
  ```

  Le funzioni del carattere \ (barra retroversa) come carattere escape nei file proprietà. Specificare una barra retroversa della costante letterale utilizzando due barre retroverse (\\).

  Modificare questo esempio di un file jt400.properties in relazione al proprio ambiente.

- **Utilizzare una classe Properties**

  Alcuni browser non caricano i file proprietà senza modificare esplicitamente le impostazioni della sicurezza. Tuttavia, molti browser consentono le proprietà nei file.class, in questo modo le proprietà di sistema IBM Toolbox per Java possono essere specificate anche da una classe definita com.ibm.as400.access.Properties che estende java.util.Properties.

  Ad esempio, per impostare la proprietà com.ibm.as400.access.AS400.proxyServer su hqoffice, utilizzare il seguente codice Java:

  ```
  package com.ibm.as400.access;

  public class Properties
  extends java.util.Properties
  {
        public Properties ()
     {
        put ("com.ibm.as400.access.AS400.proxyServer", "hqoffice");
     }
  }
  ```

  Modificare e compilare questo esempio di file sorgente Properties.java per il proprio ambiente.

Se una proprietà di sistema IBM Toolbox per Java viene impostata utilizzando più di uno dei meccanismi descritti precedentemente, la procedura è la seguente (in ordine di precedenza decrescente):

1. La proprietà di sistema viene impostata programmaticamente utilizzando java.lang.System.setProperties()
2. La proprietà di sistema viene impostata utilizzando l'opzione -D del comando java
3. La proprietà di sistema viene impostata utilizzando la classe Proprietà
4. La proprietà di sistema viene impostata utilizzando un file jt400.properties

IBM Toolbox per Java supporta le seguenti proprietà di sistema:

- Proprietà del server proxy
- Proprietà di traccia

- Proprietà CommandCall/ProgramCall
- Proprietà FTP
- Proprietà di collegamento

Proprietà server proxy

Proprietà del server proxy	Descrizione
com.ibm.as400.access.AS400.proxyServer	Specifica il nome host del server proxy e il numero porta, utilizzando il formato: `hostName:portNumber` Il numero porta è facoltativo.
com.ibm.as400.access.SecureAS400.proxyEncryptionMode	Specifica quale parte del flusso di dati proxy viene codificata utilizzando l'SSL. I valori validi sono: • 1 = Client proxy sul server proxy • 2 = Server proxy su iSeries • 3 = Client proxy sul server proxy e il server proxy su iSeries
com.ibm.as400.access.TunnelProxyServer.clientCleanupInterval	Specifica la frequenza, in secondi, con cui il server proxy effettua la ricerca di collegamenti inattivi. Il server proxy avvia un sottoprocesso per trovare i client che non comunicano più. Utilizzare questa proprietà per impostare la frequenza con cui il sottoprocesso cerca collegamenti inattivi.
com.ibm.as400.access.TunnelProxyServer.clientLifetime	Specifica il tempo, espresso in secondi, in cui un client può rimanere inattivo prima che il server proxy rimuova i riferimenti agli oggetti, in modo che la JVM raccolga i dati inutili. Il server proxy avvia un sottoprocesso per trovare i client che non comunicano più. Utilizzare questa proprietà per impostare il tempo di inattività del client prima che avvenga la raccolta di dati inutili.

Proprietà traccia

Proprietà di traccia	Descrizione
com.ibm.as400.access.Trace.category	Specifica le categorie di traccia da abilitare. Questo è un elenco delimitato da virgola che contiene tutte le combinazioni delle categorie di traccia. L'elenco completo delle categorie di traccia viene definito nella classe Trace.
com.ibm.as400.access.Trace.file	Specifica il file su cui viene scritta l'emissione della traccia. Il valore predefinito consiste nella scrittura dell'emissione traccia su System.out.
com.ibm.as400.access.ServerTrace.JDBC	Specifica le categorie di traccia da avviare sul lavoro del server JDBC. Per ulteriori informazioni sui valori supportati, consultare la Proprietà di traccia server JDBC.

Proprietà CommandCall/ProgramCall

Proprietà CommandCall/ProgramCall	Descrizione
com.ibm.as400.access.CommandCall.threadSafe	Specifica se si deve presupporre che le CommandCall siano thread-safe. Se impostato su `true`, si presuppone che le CommandCall siano thread-safe. Se impostato su `false`, si presuppone che le CommandCall non siano thread-safe. Questa proprietà viene ignorata per un dato oggetto CommandCall se è stato eseguito `CommandCall.setThreadSafe(true/false)` o `AS400.setMustUseSockets(true)` sull'oggetto.
com.ibm.as400.access.ProgramCall.threadSafe	Specifica se si deve presupporre che le ProgramCall siano thread-safe. Se impostato su `true`, si presume che le ProgramCall siano thread-safe. Se impostato su `false`, si presuppone che tutte le ProgramCall non siano thread-safe. Questa proprietà viene ignorata per un dato oggetto ProgramCall se è stato eseguito`ProgramCall.setThreadSafe(true/false)` o `AS400.setMustUseSockets(true)` sull'oggetto.

Proprietà FTP

Proprietà FTP	Descrizione
com.ibm.as400.access.FTP.reuseSocket	Specifica se il socket viene riutilizzato per più trasferimenti di file (tramite una singola istanza FTP), quando si trova in modalità attiva. Se impostato su `true`, il socket viene riutilizzato. Se impostato su `false`, per ciascun trasferimento file viene creato un nuovo socket. Questa proprietà viene ignorata per un determinato oggetto FTP se `FTP.setReuseSocket(true/false)` è stato eseguito sull'oggetto.

Proprietà di collegamento

Proprietà di collegamento	Descrizione
com.ibm.as400.access.AS400.signonHandler	Specifica l'handler di collegamento predefinito. Questa proprietà viene ignorata per un determinato oggetto AS400 se `AS400.setSignonHandler()` è stato eseguito sull'oggetto oppure se `AS400.setDefaultSignonHandler()` è stato richiamato.

Esempio: file delle proprietà

```
#=========================================================#
# IBM Toolbox per Java                                  #
#---------------------------------------------------------#
# File di esempio delle proprietà                         #
#                                                         #
# Denominare questo file jt400.properties e memorizzarlo  #
# nell'indirizzario com/ibm/as400/access a cui punta      #
# il classpath.                                          #
#=========================================================#

#---------------------------------------------------------#
# Proprietà del sistema di server proxy                   #
#---------------------------------------------------------#

# Questa proprietà di sistema specifica il nome host e il numero porta del
```

```
# server proxy, specificato nel formato: hostName:portNumber
# Il numero porta è facoltativo.
com.ibm.as400.access.AS400.proxyServer=hqoffice

# Questa proprietà di sistema specifica quale parte del flusso di dati del
# proxy viene codificato tramite SSL. I valori validi sono:
#   1 - Client proxy nel server proxy
#   2 - Server proxy nell'AS/400
#   3 - Client proxy nel proxy e server proxy nell'AS/400
com.ibm.as400.access.SecureAS400.proxyEncryptionMode=1

# Questa proprietà di sistema specifica la frequenza, in secondi,
# con cui il server proxy ricerca collegamenti inattivi. Il
# server proxy avvia un sottoprocesso per ricercare i client che
# non comunicano più. Utilizzare questa proprietà per impostare la
# frequenza con cui il sottoprocesso ricerca i collegamenti inattivi.
com.ibm.as400.access.TunnelProxyServer.clientCleanupInterval=7200

# Questa proprietà di sistema specifica il lasso di tempo, in secondi, durante
# il quale il client può rimanere inattivo prima che venga eliminato. Il server proxy
# avvia un sottoprocesso per ricercare i client che non comunicano più.
# Utilizzare questa proprietà per impostare il lasso di tempo durante il quale un
# client può rimanere inattivo prima di essere eliminato.
com.ibm.as400.access.TunnelProxyServer.clientLifetime=2700


#-------------------------------------------------------#
# Proprietà del sistema di traccia                      #
#-------------------------------------------------------#

# Questa proprietà di sistema specifica quali categorie di traccia sono abilitate.
# Questo è un elenco delimitato da una virgola contenente una combinazione delle
# categorie di traccia. L'elenco completo delle categorie di traccia è definita
# nella classe Trace.
com.ibm.as400.access.Trace.category=error,warning,information

# Questa proprietà di sistema specifica il file in cui viene scritta l'emissione
# di traccia. Il valore predefinito consiste nella scrittura dell'emissione traccia su System.out.
com.ibm.as400.access.Trace.file=c:\\temp\\trace.out


#-------------------------------------------------------#
# proprietà di sistema Command Call                     #
#-------------------------------------------------------#

# Questa proprietà di sistema specifica se CommandCalls deve presupporre
# siano thread-safe. Se impostato su true, si presuppone che tutte le CommandCall
# siano thread-safe. Se impostato su false, si presuppone che tutte le CommandCall
# non siano thread-safe. Questa proprietà viene ignorata
# per un oggetto CommandCall specificato se
# CommandCall.setThreadSafe(true/false) o
# AS400.setMustUseSockets(true) sono stati eseguiti sull'oggetto.
com.ibm.as400.access.CommandCall.threadSafe=true


#-------------------------------------------------------#
# Proprietà di sistema Program Call                     #
#-------------------------------------------------------#

# Questa proprietà di sistema specifica se si dovrebbe presupporre che le ProgramCall
# siano thread-safe. Se impostato su true, si presuppone che tutte le ProgramCall
# siano thread-safe. Se impostato su false, si presuppone che tutte le ProgramCall
# non siano thread-safe. Questa proprietà viene ignorata
# per un dato oggetto ProgramCall se
# ProgramCall.setThreadSafe(true/false) o
# AS400.setMustUseSockets(true) sono stati eseguiti sull'oggetto.
com.ibm.as400.access.ProgramCall.threadSafe=true
```

```
#-----------------------------------------------------------#
# Proprietà del sistema FTP                                 #
#-----------------------------------------------------------#

# Questa proprietà di sistema specifica se il socket viene riutilizzato
# per più trasferimenti di file (tramite una singola istanza FTP),
# quando è in modalità attiva.
# Se impostato su true, il socket viene riutilizzato. Se impostato su false,
# viene creato un nuovo socket per ciascun trasferimento di file.
# Questa proprietà viene ignorata per un determinato oggetto FTP se
# FTP.setReuseSocket(true/false) è stato eseguito sull'oggetto.
com.ibm.as400.access.FTP.reuseSocket=true


#-----------------------------------------------------------#
# Proprietà del sistema di collegamento                     #
#-----------------------------------------------------------#

# Questa proprietà di sistema specifica l'handler di collegamento predefinito.
# Questa proprietà viene ignorata per un determinato oggetto AS400 se
# AS400.setSignonHandler() è stato eseguito
# sull'oggetto o se AS400.setDefaultSignonHandler()
# è stato richiamato.
com.ibm.as400.access.AS400.signonHandler=mypackage.MyHandler

# Fine
```

Esempio: file sorgente classe proprietà di sistema

```
//=========================================================
// IBM Toolbox per Java
//---------------------------------------------------------
// File sorgente della classe proprietà di esempio
//
// Compilare questo file sorgente e memorizzare il file classe nel
// classpath.

//=========================================================
package com.ibm.as400.access;

public class Properties
extends java.util.Properties
{
      public Properties ()
   {
      /*---------------------------------------------------------*/
            /* Proprietà  del sistema di server proxy            */
      /*---------------------------------------------------------*/

            // Questa proprietà di sistema specifica il nome host e il numero porta del
            // server proxy, specificato nel formato: hostName:portNumber
            // Il numero porta è facoltativo.
      put ("com.ibm.as400.access.AS400.proxyServer", "hqoffice");

      // Questa proprietà di sistema specifica quale parte del flusso di dati del
      // proxy viene codificato tramite SSL.  I valori validi sono:
      //   1 - Client proxy nel server proxy
      //   2 - Server proxy nel server iSeries
      //   3 - Client proxy nel proxy e server proxy nel server iSeries
      put("com.ibm.as400.access.SecureAS400.proxyEncryptionMode", "1");

      // Questa proprietà di sistema specifica la frequenza, in secondi,
      // con cui il server proxy ricerca collegamenti inattivi. Il
      // server proxy avvia un sottoprocesso per ricercare i client che
      // non comunicano più. Utilizzare questa proprietà per impostare la
      // frequenza con cui il sottoprocesso ricerca i collegamenti inattivi.
```

```
put("com.ibm.as400.access.TunnelProxyServer.clientCleanupInterval", "7200");

// Questa proprietà di sistema specifica il lasso di tempo, in secondi, durante
// il quale il client può rimanere inattivo prima che venga eliminato. Il server proxy
// avvia un sottoprocesso per ricercare i client che non comunicano più.
// Utilizzare questa proprietà per impostare il lasso di tempo durante il quale un
// client può rimanere inattivo prima di essere eliminato.

put("com.ibm.as400.access.TunnelProxyServer.clientLifetime", "2700");

/*-----------------------------------------------------------*/
      /* Proprietà sistema di traccia                         */
/*-----------------------------------------------------------*/

// Questa proprietà di sistema specifica quali categorie di traccia sono abilitate.
// Questo è un elenco delimitato da una virgola contenente una combinazione delle
// categorie di traccia. L'elenco completo delle categorie di traccia è definito
// nella classe Trace.
put ("com.ibm.as400.access.Trace.category", "error,warning,information");

// Questa proprietà di sistema specifica il file in cui viene scritta l'emissione
// di traccia.  Il valore predefinito consiste nella scrittura dell'emissione traccia su System.out.
put ("com.ibm.as400.access.Trace.file", "c:\temp\trace.out");


/*-----------------------------------------------------------*/
      /* Proprietà di sistema chiamata al comando             */
/*-----------------------------------------------------------*/

// Questa proprietà di sistema specifica se si deve presupporre che le CommandCall
// siano thread-safe. Se impostato su true, si presuppone che tutte le CommandCall
// siano thread-safe. Se impostato su false, si presuppone che tutte le CommandCall
// non siano thread-safe. Questa proprietà viene ignorata
// per un oggetto CommandCall specificato se
// CommandCall.setThreadSafe(true/false) o
// AS400.setMustUseSockets(true) sono stati eseguiti sull'oggetto.
put ("com.ibm.as400.access.CommandCall.threadSafe", "true");


/*-----------------------------------------------------------*/
      /* Proprietà di sistema Program Call                   */
/*-----------------------------------------------------------*/

// Questa proprietà di sistema specifica se si dovrebbe presupporre che le ProgramCall
// siano thread-safe. Se impostato su true, si presuppone che tutte le ProgramCall
// siano thread-safe. Se impostato su false, si presuppone che tutte le ProgramCall
// non siano thread-safe. Questa proprietà viene ignorata
// per un dato oggetto ProgramCall se
// ProgramCall.setThreadSafe(true/false) o
// AS400.setMustUseSockets(true) sono stati eseguiti sull'oggetto.
put ("com.ibm.as400.access.ProgramCall.threadSafe", "true");

/*-----------------------------------------------------------*/
/* Proprietà del sistema FTP                                 */
/*-----------------------------------------------------------*/

// Questa proprietà di sistema specifica se il socket viene riutilizzato
// per più trasferimenti di file (tramite una singola istanza FTP),
// quando si trova in modalità attiva. Se impostato su true, il socket viene riutilizzato.
// Se impostato su false, per ciascun trasferimento file viene creato un nuovo socket.
// Questa proprietà viene ignorata per un determinato oggetto FTP se
// FTP.setReuseSocket(true/false) è stato eseguito sull'oggetto.
put ("com.ibm.as400.access.FTP.reuseSocket", "true");
```

```
      /*-------------------------------------------------------*/
      /* Proprietà del sistema di collegamento                 */
      /*-------------------------------------------------------*/

      // Questa proprietà di sistema specifica l'handler di collegamento predefinito.
      // Questa proprietà viene ignorata per un determinato oggetto AS400 se
      // AS400.setSignonHandler() è stato eseguito
      // sull'oggetto o se AS400.setDefaultSignonHandler()
      // è stato richiamato.
      put ("com.ibm.as400.access.AS400.signonHandler", "mypackage.MyHandler");
   }
}
```

Classi IBM Toolbox per Java

Le classi IBM Toolbox per Java vengono catalogate (come tutte le classi Java) in pacchetti. Ogni pacchetto fornisce un certo tipo di funzionalità. Per comodità, questa documentazione attribuisce, in genere, ad ogni pacchetto un nome breve. Ad esempio, il pacchetto com.ibm.as400.access viene chiamato pacchetto Access.

Utilizzare i collegamenti nel seguente elenco per trovare informazioni sulle classi nei differenti pacchetti IBM Toolbox per Java:

- Classi Access consentono di accedere e gestire le risorse su iSeries
- "Classi Commtrace" a pagina 178 consente di gestire i dati traccia delle comunicazioni per le descrizioni delle linee Ethernet o Token ring
- Classi HTML consentono di creare velocemente moduli e tabelle HTML
- Classi Micro consentono di creare i programmi Java che forniscono all'unità senza fili un accesso diretto ai dati e ai servizi del server iSeries
- Classi ReportWriter consentono di creare documenti formattati da sorgenti di dati XML
- Classi Resource utilizzano una framework comune per accedere e gestire le risorse di iSeries
- Classi Security proteggono i collegamenti con il server e verificano l'identità di un utente che lavora sul server iSeries
- Classi Servlet assistono nel richiamo e nella formattazione dei dati per l'utilizzo dei servlet Java
- Classi Utility consentono di effettuare attività amministrative come, ad esempio, le classi AS400ToolboxInstaller e AS400JarMaker
- Classi Vaccess consentono di presentare e gestire visivamente i dati

Classi Access

Le classi Access (di accesso) IBM Toolbox per Java rappresentano dati e risorse iSeries. Le classi gestiscono i server iSeries per fornire un'interfaccia abilitata ad internet per accedere ed aggiornare dati e risorse del server.

Le seguenti classi forniscono l'accesso alle risorse sul server:

- AS400 - gestisce le informazioni sul collegamento, crea e mantiene i collegamenti socket e invia e riceve dati
- SecureAS400 - abilita all'uso di un oggetto AS400 quando si inviano o si ricevono dati codificati
- AS400JPing - consente al programma Java di interrogare i server host per vedere quali servizi sono in funzione e quali porte sono in servizio
- BidiTransform - abilita a realizzare le conversioni di un testo bidirezionale

- Classi della tabella ad accesso casuale suddivise in cluster - abilitano il programma Java a condividere e replicare i dati non persistenti tra i nodi nelle tabelle ad accesso casuale divise in cluster maggiormente disponibili
- Chiamata al comando - esegue comandi batch iSeries
- Lotto di collegamenti - un lotto di oggetti di AS400, utilizzato per condividere collegamenti e gestire il numero di collegamenti che un utente può avere in un server iSeries
- Area dati - crea, accede e cancella aree di dati
- Conversione e descrizione dati - converte e gestisce dati e descrive il formato record di un buffer di dati
- Code dati - crea, accede, modifica e cancella code dati
- Certificati digitali - gestisce certificati digitali sui server iSeries
- Variabile d'ambiente - gestisce variabili d'ambiente iSeries
- Registrazione evento - fornisce un modo per registrare eccezioni e messaggi indipendenti dell'unità utilizzata per visualizzarli
- Eccezioni - visualizza errori quando, ad esempio, si verificano errori di unità o errori di programmazione
- FTP - fornisce una interfaccia programmabile per funzioni FTP
- IFS (Integrated File System) - accede ai file, apre file, apre flussi di immissioni ed emissioni ed elenca il contenuto degli indirizzari
- Chiamata applicazione Java - chiama un programma Java su un server iSeries in esecuzione sulla JVM iSeries
- JDBC - accede DB2 UDB per dati iSeries
- Lavori - accede ai lavori iSeries e alle registrazioni lavoro
- Messaggi - accede ai messaggi e alle code messaggi sul server iSeries
- Configurazione NetServer - accede e modifica lo stato e la configurazione del NetServer iSeries
- Autorizzazione - visualizza e modifica autorizzazioni oggetto sul server iSeries
- Stampa - gestisce le risorse stampa iSeries
- Licenza prodotti - gestisce le licenze per i prodotti iSeries
- Chiamata al programma - chiama un programma iSeries
- Nome percorso oggetto QSYS - rappresenta oggetti nell'IFS di iSeries
- Accesso al livello record - crea, legge, aggiorna e cancella file e membri iSeries
- Chiamata al programma di servizio - richiama un programma di servizio iSeries
- Stato di sistema - visualizza le informazioni sullo stato del sistema e consente l'accesso alle informazioni sul lotto di sistema
- Valori di sistema - richiama e modifica i valori del sistema e gli attributi di rete
- Traccia (stato di efficienza) - registra i punti di traccia e i messaggi di diagnostica
- Utenti e gruppi - accede agli utenti ed ai gruppi iSeries
- Spazio utente - accede agli spazi utente iSeries

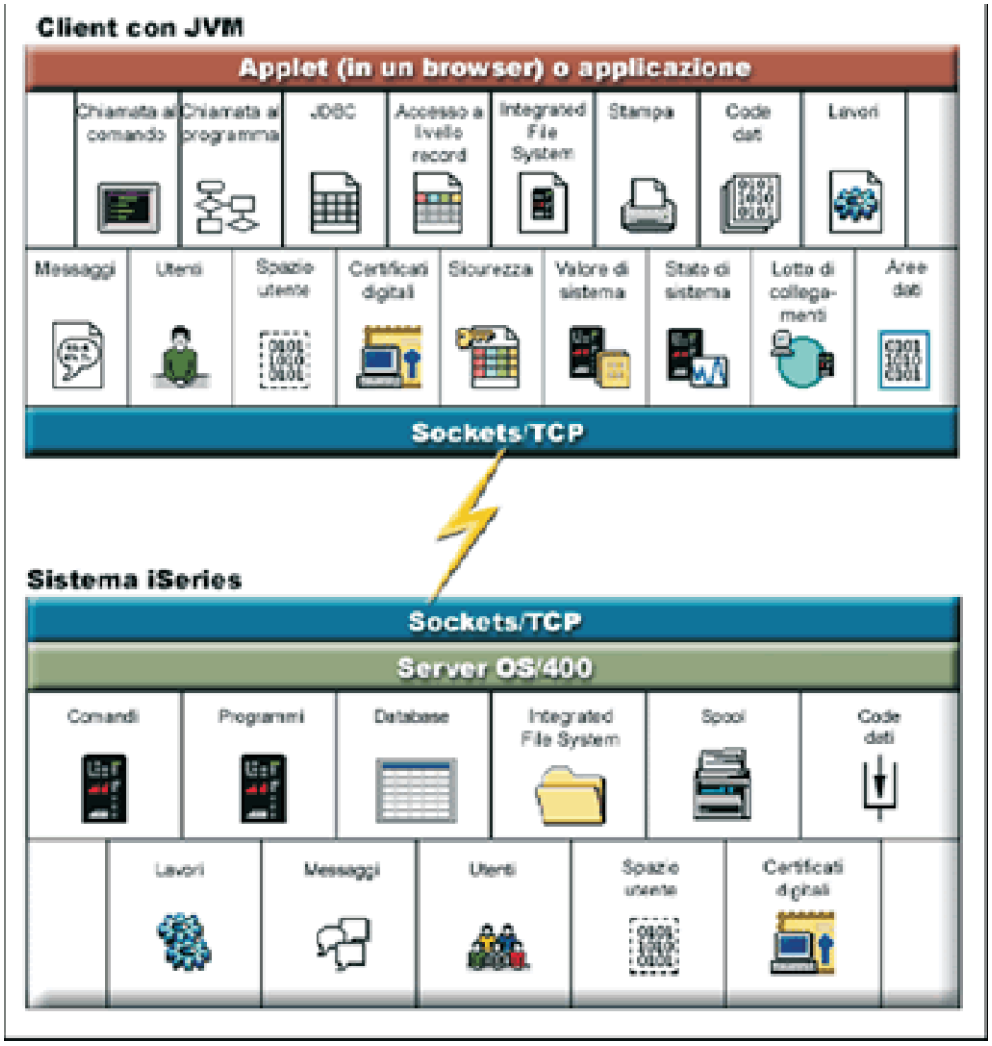
Nota: IBM Toolbox per Java fornisce una seconda serie di classi, denominate classi risorsa, per gestire gli oggetti e gli elenchi iSeries. Le classi delle risorse presentano una framework generica e un'interfaccia di programmazione coerente per gestire vari oggetti ed elenchi iSeries. Dopo aver letto informazioni relative alle classi nel pacchetto access e nel pacchetto resource, è possibile scegliere l'oggetto più adatto all'applicazione.

Punti di accesso al server

Le classi di accesso IBM Toolbox per Java forniscono una funzionalità simile all'utilizzo delle API IBM @server iSeries Access per Windows. Tuttavia, l'installazione di iSeries Access per Windows non è necessaria per l'utilizzo delle classi.

Le classi di accesso utilizzano, come punti di accesso, i server iSeries esistenti. Ogni server esegue un lavoro separato in iSeries ed invia e riceve flussi di dati su un collegamento socket.

Figura 1: punti di accesso al server



Descrizione lunga della Figura 1: punti di accesso server (rzahh501.gif): In IBM Toolbox per Java: punti di accesso server

Questa figura fornisce una sintesi grafica di come le classi nel pacchetto Access IBM Toolbox per Java utilizzano collegamenti socket per interagire con i dati e i servizi sui server iSeries.

Descrizione

La figura è composta da quanto segue:

- Un rettangolo in alto rappresenta uno o più client, ognuno con una macchina Java virtuale. L'etichetta per il client spiega che sta eseguendo un'applet in un browser o un'applicazione Java e dispone di un collegamento socket/TCP al server iSeries.
- Un rettangolo in basso rappresenta un server iSeries. L'etichetta relativa al server iSeries spiega che dispone di uno o più server su cui è in esecuzione OS/400 e tali server dispongono di una connessione socket/TCP al client.

- Un simbolo grafico a forma di fulmine collega i due rettangoli e rappresenta il collegamento socket/TCP attivo che consente alla informazioni di viaggiare tra il client e i server iSeries.

Il client (il rettangolo in alto) include le differenti funzioni nel pacchetto access di IBM Toolbox per Java che è possibile utilizzare per gestire dati e servizi sui server iSeries:

- Command call
- Chiamata al programma
- JDBC
- Accesso al livello record
- IFS (Integrated file system)
- Stampa
- Code dati
- Lavori
- Messaggi
- Utenti
- Spazio utente
- Certificati digitali
- Sicurezza
- Valore di sistema
- Stato di sistema
- Lotto di collegamenti
- Aree dati

Il server iSeries (il rettangolo in basso) include i differenti tipi di dati e servizi che è possibile gestire utilizzando classi contenute nel pacchetto access di IBM Toolbox per Java:

- Comandi
- I programmi
- Database
- IFS (Integrated file system)
- Spool
- Code dati
- Lavori
- Messaggi
- Utenti
- Spazio utente
- Certificati digitali

Classe AS400

La classe AS400 gestisce quanto segue:

- Una serie di collegamenti socket ai lavori server sul server iSeries.
- Funzionalità di collegamento per il server. Ciò include la richiesta all'utente delle informazioni sul collegamento, la memorizzazione in cache della parola d'ordine, la gestione dell'utente predefinito.

E' necessario che il programma Java fornisca un oggetto AS400 quando utilizza un'istanza di una classe che accede al server iSeries. Ad esempio, l'oggetto CommandCall richiede un oggetto AS400 prima che possa inviare i comandi al server iSeries.

L'oggetto AS400 gestisce i collegamenti, gli ID utente e le parole d'ordine in modo differente quando è in esecuzione nella JVM iSeries. Per ulteriori informazioni, consultare JVM (Java virtual machine) iSeries.

Gli oggetti AS400 supportano l'autenticazione Kerberos, utilizzando l'API (Application Programming Interface) JGSS (Java Generic Security Service) per l'autenticazione da parte del server, invece di utilizzare l'ID e la parola d'ordine.

Nota: l'utilizzo dei ticket Kerberos richiede l'installazione di J2SDK, v1.4 e la configurazione di API JGSS.

Per ulteriori informazioni su JGSS, consultare la pagina J2SDK, Documentazione di sicurezza v1.4  .

Consultare Gestione dei collegamenti per le informazioni sulla gestione dei collegamenti al server iSeries tramite l'oggetto AS400. Consultare AS400ConnectionPool per informazioni sulla riduzione del tempo di collegamento iniziale tramite la richiesta di collegamento da un lotto di collegamenti.

La classe AS400 fornisce le seguenti funzioni di collegamento:

- Autenticazione del profilo dell'utente
- Richiamo della credenziale di un token di profilo e autenticazione del profilo utente associato
- Impostazione della credenziale di un token di profilo
- Gestione degli ID utente predefiniti
- Memorizzazione delle parole d'ordine nella cache
- Richiesta dell'ID utente
- Modifica di una parola d'ordine
- Richiamo della versione e del release dell'iSeries

Per informazioni sull'utilizzo di un oggetto AS400 quando si inviano o si ricevono dati codificati, consultare Classe SecureAS400.

Gestione degli ID utente predefiniti:

Per ridurre il numero di volte in cui un utente deve collegarsi, utilizzare un ID utente predefinito. Il programma Java utilizza l'ID utente predefinito quando il programma non fornisce un ID utente. L'ID utente predefinito può essere impostato dal programma Java o dall'interfaccia utente. Se l'ID dell'utente predefinito non è stato stabilito, la finestra di dialogo Collegamento consente all'utente di impostare l'ID utente predefinito.

Una volta determinato l'ID utente per un dato server, la finestra di dialogo Collegamento non consente più di modificarlo. Una volta creato un oggetto AS400, il programma Java può fornire l'ID e la parola d'ordine dell'utente. Quando un programma fornisce l'ID utente all'oggetto AS400, l'ID utente predefinito non viene influenzato. E' necessario che il programma imposti esplicitamente l'ID utente predefinito setUseDefaultUser() se il programma desidera impostare o modificare l'ID utente predefinito. Consultare Richiesta, ID utente predefinito e riepilogo memorizzazione in cache della parola d'ordine per ulteriori informazioni.

L'oggetto AS400 dispone di metodi per richiamare, impostare ed eliminare l'ID utente predefinito. Il programma Java può anche disabilitare l'ID utente predefinito tramite il metodo setUseDefaultUser(). Se l'elaborazione dell'ID utente predefinito è disabilitata e l'applicazione Java non fornisce un ID utente, l'oggetto AS400 richiede l'ID utente ogni qualvolta si effettui un collegamento al server iSeries.

Tutti gli oggetti AS400 che rappresentano lo stesso server iSeries in una JVM utilizzano lo stesso ID utente predefinito.

Nel seguente esempio, vengono creati due collegamenti al server utilizzando due oggetti AS400. Se l'utente ha selezionato la casella ID utente predefinito al momento del collegamento, non gli verrà richiesto l'ID al secondo collegamento.

```
                    // Creare due oggetti AS400 nello
                    // stesso iSeries.
AS400 sys1 = new AS400("mySystem.myCompany.com");
AS400 sys2 = new AS400("mySystem.myCompany.com");

                    // Avviare un collegamento al servizio di chiamata
                    // del comando. All'utente viene richiesto
                    // ID utente e parola d'ordine.
    sys1.connectService(AS400.COMMAND);

                    // Avviare un altro collegamento al
                    // servizio di chiamata del comando. All'utente non viene
                    // richiesto nulla.
    sys2.connectService(AS400.COMMAND);
```

Le informazioni sull'ID utente predefinito vengono eliminate quando l'ultimo oggetto AS400 di un server iSeries viene eliminato.

Utilizzo di una memoria cache della parola d'ordine: La memoria cache della parola d'ordine permette a IBM Toolbox per Java di salvare le informazioni sull'ID utente e sulla parola d'ordine in modo tale che all'utente non vengano chieste informazioni ogni qualvolta si effettui un collegamento. Utilizzare i metodi forniti dall'oggetto AS400 per effettuare le seguenti operazioni:

- Eliminare il contenuto della cache in cui si trova la parola d'ordine e disattivarla
- Ridurre il numero di volte in cui un utente deve immettere le informazioni per effettuare il collegamento

La memorizzazione in cache viene effettuata per tutti gli oggetti AS400 che rappresentano un server iSeries in una JVM (Java virtual machine). Java non permette di condividere informazioni tra macchine virtuali, così una parola d'ordine memorizzata in cache in una JVM non è visibile da un'altra macchina virtuale. La cache rimane vuota quando l'ultimo oggetto AS400 viene eliminato. Nella finestra di dialogo Collegamento esiste una casella di spunta che, se selezionata, consente all'utente di memorizzare la parola d'ordine nella cache. Quando viene creato un oggetto AS400, il programma Java consente di fornire l'ID e la parola d'ordine dell'utente. Le parole d'ordine fornite dai programmi di creazione non sono memorizzate nella cache.

L'oggetto AS400 fornisce metodi per eliminare il contenuto della cache della parola d'ordine e per disattivare la memorizzazione in cache della parola d'ordine. Consultare Richiesta, ID utente predefinito e riepilogo memorizzazione in cache della parola d'ordine per ulteriori informazioni.

Richiesta di ID utente e parole d'ordine: Richiedere ID utente e parole d'ordine:

- Può essere necessario quando ci si collega al server iSeries
- Può essere disattivato dal programma Java

I programmi Java possono disattivare la richiesta dell'ID utente e della parola d'ordine e le finestre messaggio visualizzate dall'oggetto AS400. Ad esempio, ciò può essere necessario quando un'applicazione viene eseguita su un gateway per conto di molti client. Se le richieste e i messaggi sono visualizzati sulla macchina gateway, l'utente non ha modo di interagire con le richieste. Questi tipi di applicazioni possono disattivare tutte le richieste, utilizzando il metodo setGuiAvailable() sull'oggetto AS400.

Consultare Richiesta, ID utente predefinito e riepilogo memorizzazione in cache della parola d'ordine per ulteriori informazioni.

Riepilogo della richiesta, di ID utente predefinito e di memorizzazione in cache della parola d'ordine:

I programmi Java possono controllare le richieste dell'ID utente e della memorizzazione in cache della parola d'ordine. Le informazioni sulla finestra di dialogo Collegamento possono essere utilizzate per impostare l'ID utente predefinito e per la memorizzazione in cache della parola d'ordine. La seguente tabella indica quando si verifica la richiesta, quali informazioni sono richiamate e quali informazioni sono impostate. Questa tabella presuppone che il programma Java consenta l'elaborazione dell'ID utente predefinito e la memorizzazione in cache della parola d'ordine e che siano state selezionate le caselle **ID utente predefinito** e **Salva parola d'ordine** sulla finestra di dialogo Collegamento.

Utilizzare questa tabella per i collegamenti del client, non per eseguire Java sul server.

Sistema fornito dal programma di creazione	**ID utente fornito dal programma di creazione**	**Parola d'ordine fornita dal programma di creazione**	**Utente predefinito stabilito**	**Parola d'ordine in memoria cache per l'ID utente**	**Risultato dell'utilizzo delle impostazioni contrassegnate**
					All'utente vengono richiesti il nome del sistema, l'ID utente e la parola d'ordine. Viene stabilito l'ID utente predefinito e la parola d'ordine viene memorizzata nella cache.
Sì					All'utente vengono richiesti l'ID e la parola d'ordine. Il nome del sistema viene visualizzato ma non può essere modificato. Viene stabilito l'ID utente predefinito e la parola d'ordine viene memorizzata nella cache.
Sì	**Sì**				All'utente viene richiesta la parola d'ordine. Viene visualizzato l'ID dell'utente che può essere modificato. Il nome del sistema viene visualizzato ma non può essere modificato. L'ID utente predefinito non viene modificato. La parola d'ordine viene memorizzata nella cache.

Sistema fornito dal programma di creazione	ID utente fornito dal programma di creazione	Parola d'ordine fornita dal programma di creazione	Utente predefinito stabilito	Parola d'ordine in memoria cache per l'ID utente	Risultato dell'utilizzo delle impostazioni contrassegnate
Sì	**Sì**	**Sì**			Nessuna richiesta. L'ID utente predefinito non viene modificato. La parola d'ordine non viene memorizzata nella cache.
			Sì		All'utente vengono richiesti il nome e la parola d'ordine del sistema. Viene visualizzato l'ID dell'utente che può essere modificato. La modifica dell'ID utente non modifica l'ID dell'utente predefinito. La parola d'ordine viene memorizzata nella cache.
Sì			**Sì**		Richiesta della parola d'ordine per l'ID utente predefinito. Viene visualizzato l'ID dell'utente che può essere modificato. Il nome del sistema viene visualizzato ma non può essere modificato. La parola d'ordine viene memorizzata nella cache.
Sì			**Sì**	**Sì**	Nessuna richiesta. Effettuare il collegamento utilizzando l'ID utente predefinito e la parola d'ordine dalla cache.

Sistema fornito dal programma di creazione	ID utente fornito dal programma di creazione	Parola d'ordine fornita dal programma di creazione	Utente predefinito stabilito	Parola d'ordine in memoria cache per l'ID utente	Risultato dell'utilizzo delle impostazioni contrassegnate
Sì	**Sì**			**Sì**	Nessuna richiesta. Effettuare il collegamento utente specificato utilizzando la parola d'ordine dalla memoria cache.
Sì	**Sì**		**Sì**	**Sì**	Nessuna richiesta. Effettuare il collegamento utente specificato utilizzando la parola d'ordine dalla memoria cache.
Sì	**Sì**	**Sì**	**Sì**		Nessuna richiesta. Effettuare il collegamento come utente specificato.

Classe SecureAS400

Quando un oggetto AS400 comunica con il server, i dati dell'utente (eccetto la parola d'ordine) vengono inviati non codificati al server.In questo modo, gli oggetti IBM Toolbox associati a un oggetto AS400 scambiano dati con il server su un normale collegamento.

Quando si desidera utilizzare IBM Toolbox per Java per scambiare dati sensibili con il server, è possibile codificare i dati utilizzando SSL (Secure Sockets Layer). Utilizzare l'oggetto SecureAS400 per designare quali dati si desidera codificare.Gli oggetti IBM Toolbox per Java associati all'oggetto SecureAS400 scambiano dati con il server su un collegamento sicuro.

Per ulteriori informazioni, consultare SSL (Secure Sockets Layer) e JSSE (Java Secure Socket Extension).

La classe SecureAS400 è una sottoclasse della classe AS400.

E' possibile impostare un collegamento server sicuro creando un'istanza di un oggetto SecureAS400 nei modi che seguono:

- SecureAS400(String systemName, String userID) richiede le informazioni di collegamento
- SecureAS400(String systemName, String userID, String password) non richiede le informazioni di collegamento

Il seguente esempio mostra all'utente come utilizzare CommandCall per inviare comandi al server iSeries che utilizza un collegamento protetto:

```
 // Creare un oggetto AS400 protetto. Questa è l'unica istruzione che cambia
 // rispetto ad un caso diverso da SSL.
SecureAS400 sys = new SecureAS400("mySystem.myCompany.com");

// Creare un oggetto chiamata al comando
CommandCall cmd = new CommandCall(sys, "myCommand");
```

```
     // Eseguire i comandi.
Un collegamento protetto è stabilito quando viene
     // eseguito il comando. Tutte le informazioni che vengono passate tra il
// client e il server vengono codificate.
     cmd.run();
```

AS400JPing

L'AS400JPing consente al programma Java di interrogare i server dell'host per vedere quali servizi sono in esecuzione e quali porte sono in servizio. Per interrogare i server da una riga comandi, usare la classe JPing.

La classe AS400JPing fornisce numerosi metodi:

- Esecuzione del ping del server
- Esecuzione del ping di uno specifico servizio sul server
- Impostazione di un oggetto PrintWriter a cui si desidera collegare le informazioni sul ping
- Impostazione della verifica del supero tempo per un'operazione ping

Esempio: utilizzo di AS400JPing

Il seguente esempio mostra come utilizzare AS400JPing in un programma Java per eseguire il ping del Servizio comando remoto iSeries:

```
AS400JPing pingObj = new AS400JPing("myAS400", AS400.COMMAND, false);
    if (pingObj.ping())
        System.out.println("SUCCESS");
                    else
        System.out.println("FAILED");
```

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per informazioni legali importanti.

Classe BidiTransform

La classe AS400BidiTransform fornisce modifiche del layout che consentono di convertire un testo bidirezionale in formato iSeries (dopo averlo prima convertito in Unicode) in testo bidirezionale in formato Java e viceversa.

La classe AS400BidiTransform consente di:

- Richiamare e impostare il CCSID del sistema
- Richiamare e impostare il tipo string dei dati iSeries
- Richiamare e impostare il tipo string dei dati Java
- Convertire i dati da un layout Java ad iSeries
- Convertire i dati da un layout iSeries a Java

Esempio: utilizzo di AS400BidiTransform

Il seguente esempio mostra come si può utilizzare la classe AS400BidiTransform per convertire un testo bidirezionale:

```
// Dati Java al layout iSeries:
AS400BidiTransform abt;
abt = new AS400BidiTransform(424);
String dst = abt.toAS400Layout("some bidirectional string");
```

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per informazioni legali importanti.

Classi ClusteredHashTable

Le classi ClusteredHashTable consentono ai programmi Java di utilizzare tabelle ad accesso casuale suddivise in cluster maggiormente disponibili per condividere e duplicare i dati nella memoria non permanente tra i nodi in un cluster. Per utilizzare le classi ClusteredHashTable, verificare che si possa utilizzare la memoria volatile per i dati. I dati duplicati non sono codificati.

Nota: le seguenti informazioni presumono che si conoscano i concetti e la terminologia comuni al cluster iSeries. Per ulteriori informazioni sui cluster e su come utilizzarli, consultare Cluster.

L'utilizzo della classe ClusteredHashTable richiede la definizione e l'attivazione di un cluster sui sistemi iSeries. E' necessario anche avviare un server della tabella ad accesso casuale suddivisa in cluster. Per ulteriori informazioni, consultare Configurazione del cluster e API della tabella ad accesso casuale suddivisa in cluster.

I parametri richiesti sono il nome del server della tabella ad accesso casuale suddivisa in cluster e l'oggetto AS400, che rappresenta il sistema che contiene il server della tabella ad accesso casuale suddivisa in cluster.

Per immagazzinare i dati in un server della tabella ad accesso casuale suddivisa in cluster, sono necessari una gestione del collegamento e una chiave:

- Quando si apre un collegamento, il server della tabella ad accesso casuale suddivisa in cluster assegna la gestione del collegamento da specificare sulle successive richieste al server della tabella ad accesso casuale suddivisa in cluster. Questa gestione del collegamento è utile solo per gli oggetti AS400 in cui si sono create istanze, è necessario quindi aprire un altro collegamento se si utilizza un oggetto AS400 differente.
- Sono necessarie la chiave di accesso e i dati di modifica nella tabella ad accesso casuale suddivisa in cluster. Le chiavi duplicate non sono supportate.

La classe ClusteredHashTable fornisce metodi che consentono di eseguire queste operazioni:

- Aprire un collegamento con il lavoro del server della tabella ad accesso casuale suddivisa in cluster
- Creare una chiave univoca per memorizzare i dati in una tabella ad accesso casuale suddivisa in cluster
- Chiudere il collegamento attivo con il lavoro del server della tabella ad accesso casuale suddivisa in cluster

Alcuni metodi nella classe ClusteredHashTable utilizzano la classe ClusteredHashTableEntry per eseguire queste operazioni:

- Richiamare una voce dalla tabella ad accesso casuale suddivisa in cluster
- Memorizzare una voce nella tabella ad accesso casuale suddivisa in cluster
- Richiamare un elenco di voci da una tabella ad accesso casuale suddivisa in cluster per tutti i profili utente

Esempio: utilizzo di ClusteredHashTable

Il seguente esempio è relativo a un server della tabella ad accesso casuale suddivisa in cluster, denominato CHTSVR01. Esso presume che siano già attivi un cluster e un server della tabella ad accesso casuale suddivisa in cluster. Esso apre un collegamento, crea un tasto, inserisce una voce utilizzando la nuova chiave nella tabella ad accesso casuale suddivisa in cluster, richiama una voce dalla tabella stessa e chiude il collegamento.

```
ClusteredHashTableEntry myEntry = null;

String myData = new String("This is my data.");
System.out.println("Data to be stored: " + myData);

AS400 system = new AS400();
```

```
ClusteredHashTable cht = new ClusteredHashTable(system,"CHTSVR01");

// Aprire un collegamento.
cht.open();

// Richiamare una chiave alla tabella hash.
byte[] key = null;
key = cht.generateKey();

// Preparare alcuni dati che si desidera memorizzare nella tabella hash.
// ENTRY_AUTHORITY_ANY_USER indica che un qualsiasi utente può accedere
// alla voce che si trova nella tabella hash sottoposta a cluster.
// DUPLICATE_KEY_FAIL indica che, se la chiave specificata già esiste,
// la richiesta ClusteredHashTable.put() non avrà esito positivo.
int timeToLive = 500;
myEntry = new ClusteredHashTableEntry(key,myData.getBytes(),timeToLive,
     ClusteredHashTableEntry.ENTRY_AUTHORITY_ANY_USER,
     ClusteredHashTableEntry.DUPLICATE_KEY_FAIL);

// Memorizzare (o collocare) la voce nella tabella hash.
cht.put(myEntry);

// Richiamare un'immissione dalla tabella hash.
ClusteredHashTableEntry output = cht.get(key);

// Terminare il collegamento.
cht.close();
```

Utilizzando la classe ClusteredHashTable, l'oggetto AS400 si collega al server. Per ulteriori informazioni, consultare Gestione dei collegamenti.

CommandCall

La classe CommandCall consente ad un programma Java di richiamare un comando iSeries non interattivo. I risultati del comando sono disponibili in un elenco degli oggetti AS400Message .

L'immissione in CommandCall è la seguente:

- Stringa comandi da eseguire
- L'oggetto AS400 che rappresenta il sistema che eseguirà il comando

La stringa comandi può essere impostata sul programma di creazione, tramite il metodo setCommand() o con il metodo run().Dopo aver eseguito il comando, il programma Java può utilizzare il metodo getMessageList() per richiamare tutti i messaggi iSeries derivanti dal comando.

Utilizzando la classe CommandCall l'oggetto AS400 si collega a iSeries. Consultare la gestione dei collegamenti per informazioni sulla gestione dei collegamenti.

Quando il programma Java e il comando del server iSeries si trovano sullo stesso server, la funzione predefinita di IBM Toolbox per Java è di cercare la sicurezza del sottoprocesso per il comando nel sistema. Se protetto dal sottoprocesso, il comando viene eseguito sul sottoprocesso. E' possibile annullare la ricerca del tempo di esecuzione specificando esplicitamente la sicurezza del sottoprocesso per il comando utilizzando il metodo setThreadSafe() .

Esempi

I seguenti esempi mostrano le varie modalità di utilizzo della classe CommandCall per eseguire differenti comandi.

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per informazioni legali importanti.

Esempio: esecuzione di un comando

Il seguente esempio mostra come utilizzare la classe CommandCall per eseguire un comando su un server iSeries:

```
                    // Creare un oggetto AS400.
     AS400 sys = new AS400("mySystem.myCompany.com");

                    // Creare un oggetto chiamata al comando.
Questo
                    // programma imposta l'esecuzione del comando
                    // in un secondo momento. Può essere impostata qui sul
                    // programma di creazione.
     CommandCall cmd = new CommandCall(sys);

                    // Eseguire il comando CRTLIB
     cmd.run("CRTLIB MYLIB");

                    // Richiamare l'elenco messaggi che
                    // contiene l'esito del
                    // comando.
     AS400Message[] messageList = cmd.getMessageList();

                    // ... elaborare l'elenco messaggi.

                    // Scollegarsi poiché l'invio dei comandi
                    // al server è terminato
     sys.disconnectService(AS400.COMMAND);
```

Esempio: esecuzione di un comando specificato dall'utente

"Esempio: utilizzo di CommandCall" a pagina 473 mostra come eseguire un comando specificato dall'utente.

ConnectionPool

Utilizzare i lotti di collegamenti per condividere i collegamenti e gestire le impostazioni (lotti) di collegamenti ad un server iSeries. Ad esempio, un'applicazione può richiamare un collegamento da un lotto, utilizzarla, quindi restituirla al lotto per riutilizzarla.

La classe AS400ConnectionPool gestisce un lotto di oggetti AS400. La classe AS400JDBCConnectionPool rappresenta un lotto di AS400JDBCConnections che possono essere utilizzati da un programma Java come parte del supporto IBM Toolbox per Java il JDBC 2.0 Optional Package API. L'interfaccia JDBC ConnectionPool è inoltre supportata in JDBC 3.0 API, collegato a Java 2 Platform, Standard Edition, versione 1.4.

Un lotto di collegamenti di entrambi i tipi tiene traccia del numero dei collegamenti che crea. Utilizzando i metodi ereditati da ConnectionPool, è possibile impostare numerose proprietà del lotto di collegamenti, inclusi:

- il numero massimo di collegamenti che possono essere terminati da un lotto
- la durata massima di un collegamento
- il tempo massimo di inattività di un collegamento

In termini di prestazioni, il collegamento al server è un'operazione dispendiosa. L'utilizzo di lotti di collegamenti può migliorare le prestazioni evitando il ripetersi dei tentativi di collegamento. Ad esempio, creare i collegamenti quando si crea il lotto di collegamenti inserendo nel lotto collegamenti attivi (preconnessi).Invece di creare nuovi collegamenti, è possibile utilizzare un lotto di collegamenti che può facilmente richiamare, utilizzare, restituire e riutilizzare gli oggetti di collegamento.

Richiamare un collegamento utilizzando un AS400ConnectionPool specificando il nome del sistema, l'ID utente, la parola d'ordine e (facoltativamente) il servizio. Per specificare il servizio al quale si desidera collegarsi, utilizzare le costanti dalla classe AS400 (FILE, PRINT, COMMAND e così via).

Dopo aver richiamato ed utilizzato il collegamento, le applicazioni restituiscono i collegamenti al lotto. E' compito di ogni applicazione rimandare i collegamenti al lotto per il riutilizzo. Quando i collegamenti non vengono restituiti al lotto, il lotto di collegamenti continua a crescere di dimensioni e i collegamenti non vengono riutilizzati.

Consultare Gestione dei collegamenti per ulteriori informazioni sulla gestione quando viene aperto un collegamento con iSeries durante l'utilizzo delle classi AS400ConnectionPool.

Esempio: utilizzo di AS400ConnectionPool

"Esempio: utilizzo di AS400ConnectionPool" a pagina 475 mostra come utilizzare nuovamente gli oggetti AS400.

DataArea

La classe DataArea è una classe di base astratta che rappresenta un oggetto area dati iSeries.Questa classe base dispone di quattro sottoclassi che supportano i seguenti tipi di dati: dati del carattere, dati decimali, dati logici e aree dati locali che contengono i dati caratteri.

Utilizzando la classe DataArea, è possibile effettuare le seguenti operazioni:

- Richiamare la dimensione dell'area dati
- Richiamare il nome dell'area dati
- Restituire l'oggetto del sistema AS400 all'area dati
- Aggiornare gliattributi dell'area dati
- Impostare il sistema dove si trova l'area dati

L'utilizzo della classe DataArea collega l'oggetto AS400 al server. Consultare Gestione dei collegamenti per informazioni sulla gestione dei collegamenti.

CharacterDataArea

La classe CharacterDataArea rappresenta un area dati sul server che contiene i dati carattere.Le aree dei dati caratteri non hanno una funzione per etichettare i dati con il CCSID appropriato; quindi, l'oggetto dell'area dati presume che i dati siano nel CCSID dell'utente. Quando si scrive, l'oggetto dell'area dati converte da una stringa(Unicode) al CCSID dell'utente prima di scrivere i dati sul server. Quando si legge, l'oggetto dell'area dati presume che i dati riguardino il CCSID dell'utente e converte da quel CCSID a Unicode prima di restituire la stringa al programma. Quando si leggono i dati dall'area dati, la quantità di dati letti è data dal numero di caratteri, non dal numero di byte.

Utilizzando la classe CharacterDataArea, è possibile effettuare le seguenti operazioni:

- Ripulire l'area dati in modo che essa contenga tutti spazi vuoti.
- Creare un'area dati carattere sul sistema utilizzando i valori di proprietà predefiniti
- Creare un'area dati carattere con attributi specifici
- Cancellare l'area dati dal sistema dove essa si trova
- Restituire il nome percorso IFS dell'oggetto rappresentato dall'area dati.
- Leggere tutti i dati che sono contenuti nell'area dati
- Leggere unaquantità specificata di dati dall'area dati partendo dallo scostamento 0 o dallo scostamento specificato
- Impostare il nome percorso completo dell'IFS dell'area dati

- Scrivere i dati all'inizio dell'area dati
- Scrivere una quantità specificata di dati all'area dati partendo dallo scostamento 0 o dallo scostamento specificato

DecimalDataArea

La classe DecimalDataArea rappresenta un'area dati sul server che contiene dati decimali.

Utilizzando la classe DecimalDataArea, è possibile effettuare le seguenti operazioni:
- Ripulire l'area dati in modo che essa contenga 0.0
- Creare un area dati decimale sul sistema che utilizza valori di proprietà predefiniti
- Creare un'area dati decimali con attributi specificati
- Cancellare l'area dati dal server dove si trova l'area dati
- Riportare il numero di cifre a destra del punto decimale nell'area dati
- Restituire il nome percorso IFS dell'oggetto rappresentato dall'area dati.
- Leggere tutti i dati che sono contenuti nell'area dati
- Impostare il nome percorso completo dell'IFS dell'area dati
- Scrivere i dati all'inizio dell'area dati

Esempio: utilizzo di DecimalDataArea Il seguente esempio mostra come creare e scrivere in un'area dati decimali:

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per informazioni legali importanti.

```
   // Stabilire un collegamento al server "My400".
AS400 system = new AS400("MyServer");
   // Creare un oggetto DecimalDataArea.
QSYSObjectPathName path = new QSYSObjectPathName("MYLIB", "MYDATA", "DTAARA");
DecimalDataArea dataArea = new DecimalDataArea(system, path.getPath());
   // Creare area dati decimali sul server utilizzando i valori predefiniti.
dataArea.create();
   // Eliminare l'area dati.
dataArea.clear();
   // Scrivere sull'area dati.
dataArea.write(new BigDecimal("1.2"));
   // Leggere dall'area dati.
BigDecimal data = dataArea.read();
   // Cancellare l'area dati dal server.
dataArea.delete();
```

LocalDataArea

La classe LocalDataArea rappresenta un'area dati locale sul server.Un'area dati locale esiste come area dati caratteri sul server, ma essa presenta alcune restrizioni alle quali si deve fare attenzione.

L'area dati locale è associata ad un lavoro e non vi si può accedere da un altro lavoro. Quindi, non è possibile creare o cancellare l'area dati locale. Quando termina il lavoro del server, l'area dati locale associata a tale lavoro viene automaticamente cancellata e l'oggetto LocalDataArea che si riferisce al lavoro non è più valido. E' necessario anche notare che le aree dati locali hanno una dimensione fissa di 1024 caratteri sul server.

Utilizzando la classe LocalDataArea, è possibile effettuare le seguenti operazioni:
- Ripulire l'area dati in modo che essa contenga tutti spazi vuoti
- Leggere tutti i dati che sono contenuti nell'area dati
- Leggere una quantità specificata di dati dall'area dati partendo dallo scostamento specificato
- Scrivere i dati all'inizio dell'area dati

- Scrivere una quantità specificata di dati nell'area dati dove il primo carattere stato scritto per lo scostamento

LogicalDataArea

La classe LogicalDataArea rappresenta un'area dati sul server che contiene dati logici.

Utilizzando la classe LogicalDataArea, è possibile effettuare le seguenti operazioni:
- Ripulire l'area dati in modo che essa contenga false
- Creare un'area dati dei caratteri sul server utilizzando i valori di proprietà predefiniti
- Creare un'area dati dei caratteri con attributi specificati
- Cancellare l'area dati dal server dove si trova l'area dati
- Restituire il nome percorso IFS dell'oggetto rappresentato dall'area dati.
- Leggere tutti i dati che sono contenuti nell'area dati
- Impostare il nome percorso completo dell'IFS dell'area dati
- Scrivere i dati all'inizio dell'area dati

DataAreaEvent

La classe DataAreaEvent rappresenta un evento dell'area dati.

E' possibile utilizzare la classe DataAreaEvent con tutte le classi DataArea. Utilizzando la classe DataAreaEvent, è possibile effettuare le seguenti operazioni:
- Richiamare l'identificativo per l'evento

DataAreaListener

La classe DataAreaListener fornisce un'interfaccia per la ricezione degli eventi area dati.

E' possibile utilizzare la classe DataAreaListener con tutte le classi DataArea. E' possibile richiamare la classe DataAreaListener quando viene effettuata una delle seguenti operazioni:
- Ripulire
- Creare
- Cancellare
- Leggere
- Scrivere

DataConversion e DataDescription

Le classi **DataConversion** forniscono la capacità di convertire i dati numerici e di carattere tra i formati iSeries e Java. La conversione può essere necessaria quando si accede ai dati iSeries da un programma Java. Le classi DataConversion supportano la conversione di vari formati numerici e tra varie code page EBCDIC e Unicode.

Le classi **DataDescription** ampliano le classi di conversione dati per convertire tutti i campi in un record con una singola chiamata di metodo. La classe RecordFormat consente al programma di descrivere i dati che compongono una DataQueueEntry, un parametro ProgramCall, un record in un file database a cui si accede tramite classi di accesso a livello record o qualsiasi buffer di dati iSeries. La classe Record consente al programma di convertire il contenuto del record ed accedere ai dati tramite nome campo o indice.

Le classi **converter** consentono una conversione veloce ed efficiente tra Java e il server iSeries. BinaryConverter effettua la conversione tra schiere di byte Java e i tipi semplici Java. CharConverter effettua la conversione tra gli oggetti stringa Java e le code page OS/400. Per ulteriori informazioni, consultare quanto segue:

Programmi di conversione

Tipi di dati

AS400DataType è un'interfaccia che definisce i metodi richiesti per la conversione dei dati. Un programma Java utilizza tipi di dati quando è necessario convertire singole parti di dati. Le classi di conversione esistono per i seguenti tipi di dati:

- Numerico
- Testo (carattere)
- Composito (numerico e testo)

Esempio: utilizzo delle classi AS400DataType

Il seguente esempio mostra come utilizzare le classi AS400DataType con ProgramCall per fornire dati ai parametri del programma e per interpretare i dati restituiti nei parametri del programma.

Esempio: utilizzo delle classi AS400DataType con ProgramCall

Conversione che specifica un formato record

L'IBM Toolbox per Java fornisce classi per l'ampliamento delle classi di tipi di dati per gestire la conversione di dati un record alla volta invece di un campo per volta. Ad esempio, supponiamo che un programma Java legga dati da una coda dati. L'oggetto coda dati restituisce una schiera di byte di dati iSeries al programma Java. Questa schiera può potenzialmente contenere molti tipi di dati iSeries. L'applicazione può convertire un campo alla volta al di fuori della schiera di byte utilizzando le classi di tipi di dati oppure il programma può creare un formato record che descriva i campi nella schiera di byte. Quel record, quindi, effettua la conversione.

La conversione del formato record può essere utile quando si stanno gestendo dei dati che derivano dalla chiamata al programma, dalla coda dati e dalle classi di accesso al livello record. L'immissione e l'emissione da queste classi sono schiere di byte che possono contenere molti campi di vari tipi. I convertitori di formato record possono facilitare la conversione di questi dati tra il formato iSeries e il formato Java.

La conversione tramite formato record utilizza tre classi:

- Le classi FieldDescription identificano un campo o parametro con un tipo di dati ed un nome.
- Una classe RecordFormat descrive un gruppo di campi.
- Una classe Record unisce la descrizione di un record (nella classe RecordFormat) ai dati effettivi.
- Una classe LineDataRecordWriter scrive un record in un OutputStream in formato dati linea

Esempi: utilizzo delle classi RecordFormatConversion

I seguenti esempi mostrano come utilizzare le classi RecordFormatConversion con le code dati:

Utilizzare le classi Record e RecordFormat per inserire i dati in una coda

Utilizzare le classi FieldDescription, RecordFormat e Record

Classi di conversione per dati numerici:

Le classi di conversione per i dati numerici convertono dati numerici dal formato utilizzato sul server iSeries (chiamato **formato server** nella seguente tabella) nel formato Java. La seguente tabella mostra i tipi supportati:

Tipo numerico	Descrizione
AS400Bin2	Converte da un numero a due byte riportato nel formato del server ad un oggetto Java Short.
AS400Bin4	Converte da un numero a quattro byte riportato nel formato del server e un oggetto Java Integer.
AS400ByteArray	Converte tra due schiere di byte. Ciò è utile perché il convertitore correttamente riempie di zeri il buffer di destinazione.
AS400Float4	Converte tra un numero a virgola mobile a quattro byte riportato nel formato server e un oggetto Java Float.
AS400Float8	Converte tra un numero a virgola mobile a otto byte riportato nel formato server e un oggetto Java Double.
AS400PackedDecimal	Converte tra un numero decimale compresso nel formato server e un oggetto Java BigDecimal.
AS400UnsignedBin2	Converte tra un numero a due byte non riportato nel formato server e un oggetto Java Integer.
AS400UnsignedBin4	Converte tra un numero a quattro byte non riportato nel formato server e un oggetto Java Long.
AS400ZonedDecimal	Converte tra un numero decimale a zonature nel formato server e un oggetto Java BigDecimal.

Esempi

I seguenti esempi, mostrano le conversioni dati che utilizzano un tipo numerico nel formato server e un int Java:

Esempio: conversione dal formato server ad un int Java

```
// Creare a buffer che contenga il tipo dati server. Supporre che il buffer
// sia riempito con dati numerici nel formato server dalle code dati,
// dalla chiamata al programma e così via.
     byte[] data = new byte[100];

// Creare un programma di conversione per questo tipo dati server.
        AS400Bin4 bin4Converter = new AS400Bin4();

// Effettuare la conversione da tipo server a oggetto Java. Il numero inizia
                       // all'inizio del buffer.
     Integer intObject = (Integer) bin4Converter.toObject(data,0);

// Estrarre il tipo di Java semplice dall'oggetto Java.
     int i = intObject.intValue();
```

Esempio: conversione da un int Java al formato server

```
// Creare un oggetto Java che contenga il valore da convertire.
     Integer intObject = new Integer(22);

// Creare un programma di conversione per il tipo dati server.
        AS400Bin4 bin4Converter = new AS400Bin4();

// Effettuare la conversione dall'oggetto Java al tipo dati server.
     byte[] data = bin4Converter.toBytes(intObject);
```

```
// Rilevare il numero di byte del buffer che sono stati riempiti
                        // con il valore server.
     int length = bin4Converter.getByteLength();
```

Conversione testo:

I dati dei caratteri sono convertiti tramite la classe AS400Text.Questa classe converte i dati dei caratteri tra una codepage EBCDIC e una serie di caratteri (CCSID) e Unicode. Quando l'oggetto AS400Text viene creato, il programma Java specifica la lunghezza della stringa da convertire e il server CCSID o di codifica. Si presume che il CCSID del programma Java sia un Unicode 13488. Il metodo toBytes() effettua la conversione dal formato Java alla schiera di byte in formato iSeries. Il metodo toObject() converte da una schiera di byte in formato iSeries nel formato Java.

La classe AS400BidiTransform fornisce le modifiche del layout che consentono la conversione del testo bidirezionale nel formato iSeries (dopo la sua conversione in Unicode) a testo bidirezionale in formato Java o dal formato Java al formato iSeries.La conversione predefinita è basata sul CCSID del lavoro. Per modificare la direzione e la forma del testo, specificare un BidiStringType.Notare che dove gli oggetti IBM Toolbox per Java eseguono la conversione internamente, come nella classe DataArea, gli oggetti dispongono di un metodo per sovrascrivere il tipo di stringa. Ad esempio, la classe DataArea dispone del metodo addVetoableChangeListener() che è possibile specificare per registrare le modifiche veto a determinate proprietà, incluso il tipo stringa.

Esempio: conversione da dati di testo

Nel seguente esempio, si presume che l'oggetto DataQueueEntry restituisca il testo in EBCDIC. Nell'esempio, si effettua la conversione dei dati EBCDIC in Unicode, in modo tale che il programma Java possa utilizzare i dati:

```
// Supporre che il lavoro di impostazione della coda dati sia stato effettuato per
// richiamare il testo dall'iSeries ed i dati siano stati
// inseriti nel seguente buffer.
int textLength = 100;
byte[] data = new byte[textLength];

// Creare un programma di conversione per il tipo dati iSeries. Notare che è stato creato un
// programma di conversione predefinito. Il programma di conversione suppone che la code page
// EBCDIC di iSeries corrisponda alla locale del client. In caso contrario,
// il programma Java può specificare esplicitamente il CCSID
// EBCDIC da utilizzare. Tuttavia, si consiglia di specificare un
// CCSID ogni volta che è possibile (consultare Note: di seguito).
AS400Text textConverter = new AS400Text(textLength)

// Nota: Facoltativamente, è possibile creare un programma di conversione per un CCSID
// specifico. Utilizzare un oggetto AS400 nel caso in cui il programma venga eseguito
// come client proxy di IBM Toolbox per Java.
int ccsid = 37;
AS400 system = ...; // AS400 object
AS400Text textConverter = new AS400Text(textLength, ccsid, system);

// Nota: E' possibile inoltre creare un programma di conversione solo con l'oggetto AS400.
// Questo programma di conversione presuppone che la code page iSeries corrisponda
// al CCSID restituito dall'oggetto AS400.
AS400Text textConverter = new AS400Text(textLength, system);

// Convertire i dati da EBCDIC a Unicode. Se la lunghezza
// dell'oggetto AS400Text è superiore al numero di
// caratteri convertiti, la Stringa risultante sarà
// riempita da spazi vuoti fino alla lunghezza specificata.
String javaText = (String) textConverter.toObject(data);
```

Classi di conversione per tipi compositi:

Le classi di conversione per tipi compositi sono le seguenti

- AS400Array - Consente al programma Java di gestire una schiera di tipi di dati.
- AS400Structure - Consente al programma Java di gestire una struttura i cui elementi sono tipi di dati.

Esempio: conversione dei tipi di dati compositi

Il seguente esempio mostra la conversione da una struttura Java a una schiera di byte e viceversa. L'esempio presuppone che lo stesso formato di dati sia utilizzato sia per inviare che per ricevere dati.

```
     // Creare una struttura di tipi di dati che corrisponda a una struttura
     // che contenga: - un numero a quattro byte
     //    - quattro byte di riempimento
     //    - un numero a otto byte
     //    - 40 caratteri
     AS400DataType[] myStruct =
     {
                new AS400Bin4(),
                new AS400ByteArray(4),
                new AS400Float8(),
                new AS400Text(40)
     };

     // Creare un oggetto di conversione utilizzando la struttura.
          AS400Structure myConverter = new AS400Structure(myStruct);

     // Creare l'oggetto Java che conserva i dati da inviare al server.
          Object[] myData =
     {
        new Integer(88),        // il numero a quattro byte
        new byte[0],            // il carattere di riempimento (car. riempimento 0 per ogg. conversione)
        new Double(23.45),      // il numero di virgola mobile a otto byte
        "This is my structure"  // la stringa di carattere
     };

     // Convertire da oggetto Java alla schiera di byte.
     byte[] myAS400Data = myConverter.toBytes(myData);


     // ... inviare la schiera di byte al server. Recuperare i dati dal
     // server.
I dati recuperati corrisponderanno inoltre a una schiera di byte.


     // Convertire i dati recuperati da iSeries al formato Java.
     Object[] myRoundTripData = (Object[])myConverter.toObject(myAS400Data,0);

     // Recuperare il terzo oggetto dalla struttura. Questo è un doppione.
          Double doubleObject = (Double) myRoundTripData[2];

     // Estrarre il tipo di Java semplice dall'oggetto Java.
          double d = doubleObject.doubleValue();
```

Classi FieldDescription:

Le classi FieldDescription consentono al programma Java di descrivere il contenuto di un campo o parametro con un tipo dati e una stringa che contiene il nome del campo. Se il programma sta gestendo dati provenienti dall'accesso a livello record, esso può inoltre specificare ogni parola chiave DDS (data definition specification) di iSeries che descrive ulteriormente il campo.

Le classi FieldDescription sono le seguenti:

- BinaryFieldDescription
- CharacterFieldDescription

- DateFieldDescription
- DBCSEitherFieldDescription
- DBCSGraphicFieldDescription
- DBCSOnlyFieldDescription
- DBCSOpenFieldDescription
- FloatFieldDescription
- HexFieldDescription
- PackedDecimalFieldDescription
- TimeFieldDescription
- TimestampFieldDescription
- ZonedDecimalFieldDescription

Esempio: creazione delle descrizioni campo

Il seguente esempio presume che le voci in una coda dati abbiano lo stesso formato. Ogni voce ha un numero di messaggio (AS400Bin4), una registrazione data/ora (8 caratteri) e un testo di messaggio (50 caratteri):

```
// Creare una descrizione campo per i dati numerici. Notare che utilizza
// il tipo dati AS400Bin4. Inoltre, fornisce un nome al campo affinché
// vi si possa accedere tramite nome nella classe record.
BinaryFieldDescription bfd = new BinaryFieldDescription(new AS400Bin4(), "msgNumber");

// Creare una descrizione campo per i dati carattere. Notare che utilizza
// il tipo dati AS400Text. Inoltre, fornisce un nome al campo affinché
// la classe record possa accedervi tramite nome.
CharacterFieldDescription cfd1 = new CharacterFieldDescription(new AS400Text(8), "msgTime");

// Creare una descrizione campo per i dati carattere. Notare che utilizza
// il tipo dati AS400Text. Inoltre, fornisce un nome al campo affinché
// la classe record possa accedervi tramite nome.
CharacterFieldDescription cfd2 = new CharacterFieldDescription(new AS400Text(50), "msgText");
```

E' ora possibile raggruppare le descrizioni campo in un'istanza della classe RecordFormat. Per conoscere le modalità di aggiunta delle descrizioni campo a un oggetto RecordFormatect, consultare l'esempio nella pagina seguente:

"Classe RecordFormat"

Classe RecordFormat:

La classe RecordFormat permette al programma Java di descrivere un gruppo di campi o parametri.Un oggetto record contiene i dati descritti da un oggetto RecordFormat. Se il programma utilizza le classi di accesso al livello di record, la classe RecordFormat consente anche al programma di specificare le descrizioni per i campi chiave.

Un oggetto RecordFormat contiene una serie di descrizioni di campo. E' possibile accedere alla descrizione di campo tramite l'indice o il nome. Esistono metodi nella classe RecordFormat per eseguire le seguenti operazioni:

- Aggiungere descrizioni di campo al formato record.
- Aggiungere descrizioni del campo chiave al formato record.
- Richiamare descrizioni del campo dal formato record tramite indice o nome.
- Richiamare descrizioni del campo chiave dal formato record tramite indice o nome.
- Richiamare i nomi dei campi che costituiscono il formato record.
- Richiamare i nomi dei campi chiave che costituiscono il formato record.

- Richiamare il numero di campi nel formato record.
- Richiamare il numero di campi chiave nel formato record.
- Creare un oggetto Record basato su questo formato record.

Esempio: aggiunta delle descrizioni campo a un formato record

Il seguente esempio, mostra come aggiungere le descrizioni campo create nell'esempio descrizione campo a un formato record:

```
// Creare un oggetto formato record, quindi riempirlo con le descrizioni campo.
     RecordFormat rf = new RecordFormat();
     rf.addFieldDescription(bfd);
     rf.addFieldDescription(cfd1);
     rf.addFieldDescription(cfd2);
```

Per conoscere le modalità di creazione di un record da un formato record, vedere l'esempio nella pagina seguente:

"Classe Record"

Classe Record:

La classe record consente al programma Java di elaborare i dati descritti dalla classe Record Format.I dati sono convertiti tra le schiere di byte che contengono i dati del server e gli oggetti Java. Nella classe Record sono forniti i metodi per eseguire queste operazioni:

- Richiamare il contenuto di un campo, per indice o per nome, come un oggetto Java.
- Richiamare il numero di campi nel record.
- Impostare il contenuto di un campo, per indice o per nome, con un oggetto Java.
- Richiamare il contenuto del record come dati del server in una schiera di byte o flusso di emissioni.
- Impostare il contenuto del record da una schiera di byte o un flusso di immissioni.
- Convertire il contenuto del record in una stringa.

Esempio: lettura di un record

Il seguente esempio utilizza il formato record creato nell'esempio formato record:

```
// Supporre che il lavoro di impostazione della coda dati sia stato effettuato. Leggere un
// record dalla coda dati.
     DataQueueEntry dqe = dq.read();

// I dati dalla coda dati si trovano ora in una voce coda dati. Recuperare
// i dati della vice coda dati e inserirli nel record.
// In questo modo, si ottiene un record predefinito dall'oggetto formato record
// ed è possibile inizializzarlo con i dati dalla voce coda dati.
     Record dqRecord = rf.getNewRecord(dqe.getData());

// Ora che i dati si trovano nel record, recuperare i dati da un campo
// alla volta, convertendo i dati non appena vengono rimossi. Il risultato sarà
// la presenza di dati in un oggetto Java che il programma potrà elaborare.
     Integer msgNumber = (Integer) dqRecord.getField("msgNumber");
     String  msgTime   = (String)  dqRecord.getField("msgTime");
     String  msgText   = (String)  dqRecord.getField("msgText");
```

Richiamo del contenuto di un campo:

Richiamare il contenuto di un oggetto Record richiamando, tramite il programma Java, un campo alla volta oppure tutti i campi nello stesso momento. Utilizzare il metodo getField() per richiamare un singolo campo per nome o per indice. Utilizzare il metodo getFields() per richiamare tutti i campi come Oggetto[].

Il programma Java deve assegnare l'Oggetto (o l'elemento dell'Oggetto[]) restituito all'appropriato oggetto Java per il campo richiamato. La seguente tabella mostra l'appropriato oggetto Java da assegnare in base al tipo di campo.

Tipo di campo (DDS)	Tipo di campo (FieldDescription)	Oggetto Java
BINARY (B), length <= 4	BinaryFieldDescription	Short
BINARY (B), length >= 5	BinaryFieldDescription	Numero intero
CHARACTER (A)	CharacterFieldDescription	Stringa
DBCS Either (E)	DBCSEitherFieldDescription	Stringa
DBCS Graphic (G)	DBCSGraphicFieldDescription	Stringa
DBCS Only (J)	DBCSOnlyFieldDescription	Stringa
DBCS Open (O)	DBCSOpenFieldDescription	Stringa
DATE (L)	DateFieldDescription	Stringa
FLOAT (F), single precision	FloatFieldDescription	Float
FLOAT (F), double precision	FloatFieldDescription	Double
HEXADECIMAL (H)	HexFieldDescription	byte[]
PACKED DECIMAL (P)	PackedDecimalFieldDescription	BigDecimal
TIME (T)	TimeDecimalFieldDescription	Stringa
TIMESTAMP (Z)	TimestampDecimalFieldDescription	Stringa
ZONED DECIMAL (P)	ZonedDecimalFieldDescription	BigDecimal

Impostazione del contenuto di un campo:

Impostare il contenuto di un oggetto Record utilizzando il metodo setField() nel programma Java.E' necessario che il programma Java specifichi l'appropriato oggetto Java per il campo impostato. La seguente tabella mostra l'appropriato oggetto Java per ogni possibile tipo di campo.

Tipo di campo (DDS)	Tipo di campo (FieldDescription)	Oggetto Java
BINARY (B), length <= 4	BinaryFieldDescription	Short
BINARY (B), length >= 5	BinaryFieldDescription	Numero intero
CHARACTER (A)	CharacterFieldDescription	Stringa
DBCS Either (E)	DBCSEitherFieldDescription	Stringa
DBCS Graphic (G)	DBCSGraphicFieldDescription	Stringa
DBCS Only (J)	DBCSOnlyFieldDescription	Stringa
DBCS Open (O)	DBCSOpenFieldDescription	Stringa
DATE (L)	DateFieldDescription	Stringa
FLOAT (F), single precision	FloatFieldDescription	Float
FLOAT (F), double precision	FloatFieldDescription	Double
HEXADECIMAL (H)	HexFieldDescription	byte[]
PACKED DECIMAL (P)	PackedDecimalFieldDescription	BigDecimal
TIME (T)	TimeDecimalFieldDescription	Stringa
TIMESTAMP (Z)	TimestampDecimalFieldDescription	Stringa
ZONED DECIMAL (P)	ZonedDecimalFieldDescription	BigDecimal

Classe LineDataRecordWriter:

La classe LineDataRecordWriter scrive i dati record, in formato dati linea, su un OutputStream.La classe converte i dati in byte utilizzando il CCSID specificato. Il formato record associato al record determina il formato dei dati.

L'utilizzo di LineDataRecordWriter richiede che siano impostati i seguenti attributi di formato record:

- ID del formato Record
- Tipo di formato record

In congiunzione con le classi Record o le classi RecordFormat, il LineDataRecordWriter considera un record come immissione nel metodo writeRecord().(Il record considera un RecordFormat come immissione quando si creano istanze ad esso relative.)

La classe LineDataRecordWriter fornisce metodi che consentono di:

- Richiamare il CCSID
- Richiamare il nome della codifica
- Scrivere i dati record, in formato dati linea, in un OutputStream

Esempio: utilizzo della classe LineDataRecordWriter

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per informazioni legali importanti.

Il seguente esempio mostra un modo per utilizzare la classe LineDataRecordWriter per scrivere un record:

```
  // Esempio di utilizzo della classe LineDataRecordWriter.
                            try
{
          // creare un ccsid
          ccsid_ = system_.getCcsid();

          // creare una coda di emissione e specificare i dati del file di spool in modo che siano *LINE
  OutputQueue outQ = new OutputQueue(system_, "/QSYS.LIB/RLPLIB.LIB/LDRW.OUTQ");
               PrintParameterList parms = new PrintParameterList();
  parms.setParameter(PrintObject.ATTR_PRTDEVTYPE, "*LINE");

          // inizializzare il formato record per la scrittura dei dati
          RecordFormat recfmt = initializeRecordFormat();

          // creare un record ed assegnare dati da stampare...
          Record record = new Record(recfmt);
          createRecord(record);

          SpooledFileOutputStream os = null;
  try {
                  // creare il file di spool di emissione per contenere i dati del record
                  os = new SpooledFileOutputStream(system_, parms, null, outQ);
  }
          catch (Exception e) {
      System.out.println("Error occurred creating spooled file");
              e.printStackTrace();
  }

                  // creare il line data record writer
              LineDataRecordWriter ldw;
                  ldw = new LineDataRecordWriter(os, ccsid_, system_);

            // scrivere il record di dati
                  ldw.writeRecord(record);

            // chiudere outputstream
                  os.close();
}
```

```
  catch(Exception e)
{
  failed(e, "Exception occurred.");
}
```

DataQueue

Le classi DataQueue consentono al programma Java di interagire con le code dati del server. Le code dati sui server iSeries dispongono delle seguenti caratteristiche:

- La coda dati consente comunicazioni rapide tra lavori. Pertanto, essa costituisce un modo eccellente per sincronizzare e trasmettere dati tra i lavori.
- Molti lavori possono accedere simultaneamente alle code dati
- I messaggi su una coda dati sono in formato libero. I campi non sono necessari come lo sono nei file del database.
- La coda dati può essere utilizzata sia per attività di elaborazione sincrone che asincrone.
- E' possibile ordinare i messaggi presenti sulla coda dati in uno dei seguenti modi:
  - LIFO (Last-in first-out). L'ultimo (il più recente) messaggio che si trova sulla coda dati è il primo messaggio che viene estratto dalla coda.
  - FIFO (First-in first-out). Il primo (il più vecchio) messaggio che si trova sulla coda dati è il primo messaggio che viene estratto dalla coda.
  - Con chiave. Ogni messaggio sulla coda dati ha associata una chiave. Un messaggio può essere estratto dalla coda solo specificando la chiave associata.

Le classi DataQueue forniscono una serie completa di interfacce per accedere alle code dati del server dal programma Java. Si tratta di un eccellente modo di comunicare tra i programmi Java e i programmi sul server che sono scritti in un qualsiasi linguaggio di programmazione.

Un parametro necessario ad ogni oggetto della coda dati è l'oggetto AS400 che rappresenta il server che possiede la coda dati o dove la coda dati è stata creata.

L'utilizzo delle classi DataQueue collega l'oggetto AS400 al server. Consultare la gestione dei collegamenti per informazioni sulla gestione dei collegamenti.

Ogni oggetto coda dati richiede il nome del percorso dell'IFS della coda dati. DTAQ è il tipo per la coda dati. Per ulteriori informazioni, consultare i nomi percorso IFS (Integrated file system).

Code dati con chiave e sequenziali

Le classi di coda dati supportano i seguenti tipi di code dati:

- Code dati sequenziali
- Code dati con chiave

I metodi comuni ad entrambi i tipi di code si trovano nella classe BaseDataQueue .La classe DataQueue estende la classe BaseDataQueue per completare le implementazioni di code dati sequenziali.La classe BaseDataQueue viene estesa dalla classe KeyedDataQueue per completare l'implementazione di code dati con chiave.

Durante la lettura dei dati da una coda dati, questi vengono collocati in un oggetto DataQueueEntry.Questo oggetto conserva i dati sia per le code dati con chiave che per quelle sequenziali. Dati supplementari disponibili durante la lettura da una coda dati con chiave vengono collocati in un oggettoKeyedDataQueueEntry che estende la classe DataQueueEntry.

Le classi coda dati non modificano i dati che sono scritti o letti nella coda dati del server. E' necessario che il programma Java formatti correttamente i dati. Le classi DataConversion forniscono i metodi per convertire i dati.

Esempio: utilizzo di DataQueue e DataQueueEntry

Il seguente esempio crea un oggetto DataQueue, legge i dati dall'oggetto DataQueueEntry e quindi si scollega dal sistema.

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per informazioni legali importanti.

```
                        // Creare un oggetto AS400
    AS400 sys = new AS400("mySystem.myCompany.com");

                        // Creare l'oggetto DataQueue
        DataQueue dq = new DataQueue(sys, "/QSYS.LIB/MYLIB.LIB/MYQUEUE.DTAQ");

                        // leggere i dati dalla coda
        DataQueueEntry dqData = dq.read();

                        // estrarre i dati dall'oggetto DataQueueEntry.
        byte[] data = dqData.getData();

                        // ... elaborare i dati

                        // Scollegarsi poiché è terminato l'utilizzo delle code dati
        sys.disconnectService(AS400.DATAQUEUE);
```

Code dati sequenziali:

Le voci nella coda dati sequenziali sul server vengono rimosse in sequenza FIFO (first-in first-out) o LIFO (last-in first-out). Le classi BaseDataQueue e DataQueue forniscono i seguenti metodi per gestire code dati sequenziali:

- Creare una coda dati sul server.E' necessario che il programma Java specifichi la dimensione massima di una voce nella coda dati. Il programma Java può specificare, facoltativamente, ulteriori parametri sulla coda dati (FIFO / LIFO, salva informazioni mittente, specifica informazioni autorizzazioni, forza il disco e fornisce una descrizione della coda) quando la coda viene creata.
- Effettuare il peek ad una voce nella coda dati senza rimuoverla dalla coda. Il programma Java può attendere o restituire immediatamente se nessuna voce è attualmente nella coda.
- Leggere una voce da una coda. Il programma Java può attendere o restituire immediatamente se nessuna voce è disponibile nella coda.
- Scrivere una voce nella coda.
- Cancellare tutte le voci dalla coda.
- Cancellare la coda.

La classe BaseDataQueue fornisce ulteriori metodi per richiamare gli attributi della coda dati.

Esempi: gestione delle code dati sequenziali

I seguenti esempi di code dati sequenziali mostrano in che modo il produttore inserisce gli elementi in una coda dati e come il consumatore estrae gli elementi dalla coda e li elabora:

"Esempio: utilizzo delle classi DataQueue per inserire i dati in una coda" a pagina 480

"Esempio: utilizzo delle classi DataQueue per leggere le voci da una coda dati" a pagina 483

Code dati con chiavi:

Le classi BaseDataQueue e KeyedDataQueue forniscono i seguenti metodi per gestire code dati con chiave:

- Creare una coda dati con chiave sul server.E' necessario che il programma Java specifichi la lunghezza e la dimensione massima della chiave di una voce sulla coda. Il programma Java può, facoltativamente, specificare le informazioni sull'autorizzazione, salvare le informazioni sul mittente, forzarle sul disco e fornire una descrizione della coda.
- Effettuare il peek ad una voce basata sulla chiave specificata senza rimuoverla dalla coda. Il programma Java può attendere o effettuare immediatamente la restituzione se nella coda attualmente non vi è alcuna voce che corrisponda ai criteri della chiave.
- Leggere una voce dalla coda basata sulla chiave specificata. Il programma Java può attendere o effettuare immediatamente la restituzione se nella coda attualmente non vi è alcuna voce che corrisponde ai criteri della chiave.
- Scrivere una voce con chiave nella coda.
- Eliminare tutte le voci o tutte le voci che corrispondono ad una chiave specificata.
- Cancellare la coda.

Le classi BaseDataQueue e KeyedDataQueue forniscono anche ulteriori metodi per richiamare gli attributi dalla coda dati.

Esempi: gestione delle code dati con chiavi

I seguenti esempi di code dati con chiavi mostrano in che modo il produttore inserisce gli elementi in una coda dati e come il consumatore estrae gli elementi dalla coda e li elabora:

"Esempio: utilizzo di KeyedDataQueue" a pagina 488

"Esempio: utilizzo delle classi KeyedDataQueue per leggere le voci da una coda dati." a pagina 491

Certificati digitali

I Certificati digitali sono istruzioni firmate in maniera digitale utilizzate per proteggere le transazioni su internet. (Certificati digitali possono essere utilizzati sui server che eseguono OS/400 Versione 4 Release 3 (V4R3) e versioni successive.) Per proteggere un collegamento utilizzando SSL (Secure Sockets Layer), è necessario un certificato digitale.

I Certificati digitali comprendono quanto segue:

- La chiave di codifica pubblica dell'utente
- Il nome e l'indirizzo dell'utente
- La firma digitale rilasciata dall'AC (autorità di certificazione) di terzi. La firma di autorizzazione significa che l'utente è un'entità garantita.
- La data di emissione del certificato
- La data di scadenza del certificato

Come responsabile di un server protetto, l'utente può aggiungere al server una chiave principale fidata della autorità di certificazione. Questo significa che il server autorizza chi possiede la certificazione rilasciata da quella specifica autorità di certificazione.

I certificati digitali forniscono anche la codifica, garantendo un trasferimento protetto di dati tramite una chiave di codifica privata.

E' possibile creare certificati digitali tramite il programma javakey. (Per ulteriori informazioni su javakey e sulla sicurezza Java, consultare all'indirizzo la pagina la pagina Sun Microsystems, Inc., Java Security

). Il programma su licenza IBM Toolbox per Java dispone di classi che gestiscono i certificati digitali

Le classi AS400Certificate forniscono i metodi per gestire i certificati codificati X.509 ASN.1. Le classi consentono di eseguire queste operazioni:

- Richiamare e impostare i dati del certificato.
- Elencare i certificati con l'elenco convalide o il profilo utente.
- Gestire certificati, ad esempio, aggiungere un certificato al profilo utente o cancellare un certificato da un elenco di convalide.

Utilizzando una classe di certificato si collega l'oggetto AS400 al server. Consultare la gestione dei collegamenti per informazioni sulla gestione dei collegamenti.

Sul server, i certificati appartengono ad un elenco di convalide o a un profilo utente.

- La classe AS400CertificateUserProfileUtil dispone dei metodi per la gestione dei certificati su un profilo utente.
- La classe AS400CertificateVldlUtil dispone dei metodi per la gestione dei certificati in un elenco di convalida.

Per utilizzare AS400CertificateUserProfileUtil e AS400CertificateVldlUtil è necessario installare l'opzione 34 del sistema operativo di base (Digital Certificate Manager). Queste due classi estendono AS400CertificateUtil, che è una classe di base astratta che definisce i metodi comuni ad entrambe le sottoclassi.

La classe AS400Certificate fornisce i metodi per leggere e scrivere i dati del certificato.E' possibile accedere ai dati tramite una schiera di byte. Il pacchetto Java.Security in JVM 1.2 fornisce classi che possono essere utilizzate per richiamare ed impostare i campi individuali del certificato.

Elencare i certificati

Per richiamare un elenco di certificati, il programma Java deve eseguire queste operazioni:

1. Creare un oggetto AS400.
2. Creare un adeguato oggetto del certificato. Oggetti differenti sono utilizzati per elencare i certificati su un profilo utente (AS400CertificateUserProfileUtil) contro l'elenco di certificati in un elenco di convalida (AS400CertificateVldlUtil).
3. Creare criteri di selezione basati sugli attributi del certificato. La classe AS400CertificateAttribute contiene attributi utilizzati come criteri di selezione.Uno o più oggetti attributo definiscono i criteri che devono essere soddisfatti prima che un certificato venga aggiunto all'elenco. Ad esempio, un elenco deve contenere solo i certificati di un determinato utente o di una organizzazione.
4. Creare uno spazio utente sul server ed immettervi il certificato. Un'operazione di elenco può creare grandi quantità di dati. I dati vengono immessi in uno spazio utente prima che un programma Java possa richiamarli. Utilizzare il metodo listCertificates() per immettere i certificati in uno spazio utente.
5. Utilizzare il metodo getCertificates() per richiamare i certificati dallo spazio utente.

Esempio: elenco dei certificati digitali

Il seguente esempio elenca i certificati in un elenco di convalida. Esso elenca solo quei certificati che appartengono ad una determinata persona.

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per informazioni legali importanti.

```
        // Creare un oggetto AS400.
I certificati si trovano su questo sistema.
     AS400 sys = new AS400("mySystem.myCompany.com");

                               // Creare l'oggetto certificato.
          AS400CertificateVldlUtil certificateList =
                      new AS400CertificateVldlUtil(sys, "/QSYS.LIB/MYLIB.LIB/CERTLIST.VLDL");

        // Creare l'elenco attributi del certificato. Si richiedono solo certificati
        // realtivi a una singola persona in modo tale che l'elenco sia composto da un solo elemento.
          AS400CertificateAttribute[] attributeList = new AS400CertificateAttribute[1];
     attributeList[0] =
        new AS400CertificateAttribute(AS400CertificateAttribute.SUBJECT_COMMON_NAME, "Jane Doe");

        // Richiamare l'elenco che corrisponde ai criteri. Verrà utilizzato lo spazio utente "myspace"
        // nella libreria "mylib" per la memorizzazione dei certificati.
        // E' necessario che lo spazio utente sia presente prima di richiamare questa API.
     int count = certificateList.listCertificates(attributeList, "/QSYS.LIB/MYLIB.LIB/MYSPACE.USRSPC");

        // Richiamare i certificati dallo spazio utente.
     AS400Certificates[] certificates =
        certificateList.getCertificates("/QSYS.LIB/MYLIB.LIB/MYSPACE.USRSPC", 0, 8);

        // Elaborare i certificati
```

Classe EnvironmentVariable

La classe EnvironmentVariable e la classe EnvironmentVariableList consente di accedere alle ed impostare le variabili di ambiente al **livello sistema** di iSeries.

Ogni variabile possiede identificativi univoci: il nome del sistema e il nome della variabile d'ambiente. Ogni variabile di ambiente è associata ad un CCSID, che è per impostazione predefinita il CCSID del lavoro attuale, che descrive dove è memorizzato il contenuto della variabile.

Nota: le variabili di ambiente sono diverse dai valori di sistema, anche se esse vengono spesso utilizzate per lo stesso scopo. Consultare SystemValues per ulteriori informazioni su come accedere ai valori del sistema.

Utilizzare un oggetto EnvironmentVariable per effettuare le seguenti operazioni su una variabile d'ambiente:

- Richiamare ed impostare il nome
- Richiamare ed impostare il sistema
- Richiamare ed impostare il valore (che consente di modificare il CCSID)
- Aggiornare il valore

Esempio: creazione, impostazione e richiamo di variabili d'ambiente

Il seguente esempio crea due EnvironmentVariables ed imposta e richiama i valori.

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per informazioni legali importanti.

```
        // Creare l'oggetto sistema iSeries
        AS400 system = new AS400("mySystem");

        // Creare la variabile d'ambiente colore primo piano ed impostarla sul rosso.
        EnvironmentVariable fg = new EnvironmentVariable(system, "FOREGROUND");
        fg.setValue("RED");

        // Creare la variabile d'ambiente colore sfondo e richiamare il relativo valore.
        EnvironmentVariable bg = new EnvironmentVariable(system, "BACKGROUND");
        String background = bg.getValue();
```

Eccezioni

Le classi di accesso IBM Toolbox per Java lanciano eccezioni quando si verificano errori di unità, limitazioni fisiche, errori di programmazione o errori di immissione utente. Le classi di eccezioni si basano sul tipo di errore che si verifica invece che sull'ubicazione in cui l'errore ha origine.

La maggior parte delle eccezioni contengono le seguenti informazioni:

- **Tipo di errore:** l'oggetto eccezione emesso indica il tipo di errore che si è verificato. Gli errori dello stesso tipo sono raggruppati insieme in una classe di eccezioni.
- **Dettagli dell'errore:** l'eccezione contiene un codice di ritorno utile per identificare ulteriormente le cause dell'errore che si è verificato. I valori del codice di ritorno sono costanti in una classe di eccezioni.
- **Testo errore:** L'eccezione contiene una stringa di testo che descrive l'errore che si è verificato. La stringa è convertita nella locale della JVM del client.

Esempio: come catturare un'eccezione emessa

Il seguente esempio mostra come catturare un'eccezione inviata, richiamare il codice di ritorno e visualizzare il testo di eccezione:

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per informazioni legali importanti.

```
// Viene effettuato tutto il lavoro di impostazione per cancellare un file sul server attraverso
// la classe IFSFile. Provare a cancellare il file.
                           try
{
           aFile.delete();

}

                        // Cancellazione non riuscita.
     catch (ExtendedIOException e)
{
   // Visualizzare la stringa convertita contenente il motivo della
                           // mancata cancellazione.
           System.out.println(e);

   // Richiamare il codice di ritorno dell'eccezione e visualizzare ulteriori
   // informazioni in base al codice di ritorno.
           int rc = e.getReturnCode()

           switch (rc)
   {
                 case ExtendedIOException.FILE_IN_USE:
                       System.out.println("Delete failed, file is in use "):
                                 break;

                 case ExtendedIOException.PATH_NOT_FOUND:
                       System.out.println("Delete failed, path not found ");
                                 break;

      // Per ogni errore specifico di cui si desidera tener traccia...

              default:
                       System.out.println("Delete failed, rc = ");
                       System.out.println(rc);
   }
}
```

Classe FTP

La classe FTP fornisce un'interfaccia programmabile per le funzioni FTP. Non è più necessario utilizzare java.runtime.exec() o eseguire i comandi FTP in un'applicazione differente. E' quindi possibile programmare funzioni FTP direttamente nella propria applicazione. Di conseguenza, nel proprio programma, è possibile:

- Collegarsi ad un server FTP
- Inviare comandi al server
- Elencare i file in un indirizzario
- Richiamare i file dal server **e**
- Immettere file nel server

Esempio: utilizzo di FTP per copiare i file da un server

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per informazioni legali importanti.

Ad esempio, con la classe FTP, è possibile copiare una serie di file da un indirizzario sul server:

```
FTP client = new FTP("myServer", "myUID", "myPWD");
client.cd("/myDir");
     client.setDataTransferType(FTP.BINARY);
     String [] entries = client.ls();

     for (int i = 0; i < entries.length; i++)
{
  System.out.println("Copying " + entries[i]);
                              try
  {
    client.get(entries[i], "c:\\ftptest\\" + entries[i]);
  }
                              catch (Exception e)
  {
    System.out.println("  copy failed, likely this is a directory");
  }
}

     client.disconnect();
```

FTP è un'interfaccia generica che gestisce vari server FTP. Perciò, è compito del programmatore fare in modo di adattare le semantiche al server.

Sottoclasse AS400FTP

Mentre la classe FTP è un'interfaccia FTP generica, la sottoclasse AS400FTP viene scritta specificamente per il server FTP sul server. Cioè, riconosce le semantiche del server FTP sul server iSeries. Ad esempio, questa classe riconosce le varie procedure necessarie per trasferire un file di salvataggio sul server ed esegue queste procedure automaticamente. Inoltre AS400FTP è associata alle funzioni relative alla sicurezza di IBM Toolbox per Java. Come altre classi IBM Toolbox per java, AS400FTP dipende dall'oggetto AS400 per il nome sistema, ID utente e parola d'ordine.

Esempio: utilizzo di AS400FTP per salvare un file sul server

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per informazioni legali importanti.

Nel seguente esempio viene immesso un file di salvataggio nel server. Notare che l'applicazione non imposta il tipo di trasferimento dati su binario né utilizza CommandCall per creare il file di salvataggio. Poiché l'estensione è .savf, la classe AS400FTP rileva che il file da immettere è un file di salvataggio, quindi esegue queste procedure automaticamente.

```
AS400 system = new AS400();
 AS400FTP ftp = new AS400FTP(system);
ftp.put("myData.savf", "/QSYS.LIB/MYLIB.LIB/MYDATA.SAVF");
```

IFS (Integrated file system)

Le classi IFS (integrated file system) consentono al programma Java di accedere ai file nell'IFS di un server iSeries come un flusso di byte o un flusso di caratteri. Le classi IFS sono state create perché il pacchetto java.io non fornisce la funzione di reindirizzamento file e altre funzioni iSeries.

La funzione fornita dalle classi IFSFile è una serie superiore della funzione fornita dalle classi IO file nel pacchetto java.io. Tutti i metodi in FileInputStream, FileOutputStream e RandomAccessFile di java.io si trovano nelle classi IFS.

Inoltre, le classi contengono i metodi per effettuare quanto segue:

- Specificare un metodo di condivisione file per negare accesso al file se in uso
- Specificare un metodo di creazione file per aprire, creare o sostituire il file
- Bloccare una sessione del file e negare l'accesso a quella parte di file se in uso
- Elencare i contenuti di un indirizzario in modo più efficiente
- Memorizzare in cache il contenuto di un indirizzario per migliorare le prestazioni, limitando le chiamate al server
- Determinare il numero di byte disponibili sul file system del server
- Consentire all'applet Java di accedere ai file nel file system del server
- Leggere e scrivere dati come testo invece che come dati binari
- Determinare il tipo di oggetto file (logico, fisico, di salvataggio e così via) quando l'oggetto si trova nel file system QSYS.LIB

Attraverso le classi IFS, il programma Java può accedere direttamente ai file di flusso su iSeries. Il programma Java può ancora utilizzare il pacchetto java.io, ma il sistema operativo del client deve fornire un metodo di reindirizzamento. Ad esempio, se il programma Java viene avviato sul sistema operativo Windows 95 o Windows NT, la funzione Unità di rete di iSeries Access per Windows è necessaria per reindirizzare le chiamate java.io a iSeries. Con le classi IFS, non è necessario iSeries Access per Windows.

Un parametro necessario per le classi IFS è l'oggetto AS400 che rappresenta il sistema iSeries che contiene il file. L'utilizzo delle classi IFS fa sì che l'oggetto AS400 si colleghi a iSeries. Consultare la gestione dei collegamenti per informazioni sulla gestione dei collegamenti.

Le classi IFS richiedono un nome gerarchico dell'oggetto nell'IFS. Utilizzare la barra come carattere di separazione del percorso. L'esempio che segue mostra come accedere a FILE1 nel percorso indirizzario DIR1/DIR2:

```
/DIR1/DIR2/FILE1
```

Classi IFS (Integrated file system)

La seguente tabella elenca le classi IFS:

Classe IFS	Descrizione
IFSFile	Rappresenta un file in IFS
IFSJavaFile	Rappresenta un file in IFS (estende java.io.File)
IFSFileInputStream	Rappresenta un flusso di immissioni per la lettura di dati da un file iSeries
IFSTextFileInputStream	Rappresenta un flusso di dati carattere letti da un file

Classe IFS	Descrizione
IFSFileOutputStream	Rappresenta un flusso di emissioni per la scrittura di dati su un file iSeries
IFSTextFileOutputStream	Rappresenta un flusso di dati carattere scritti su un file
IFSRandomAccessFile	Rappresenta un file su iSeries per la lettura e la scrittura di dati
IFSFileDialog	Consente di spostarsi nel file system e di selezionare un file in esso

Esempi: utilizzo delle classi IFS

"Esempio: utilizzo delle classi IFS per copiare un file da un indirizzario ad un altro." a pagina 499 mostra come utilizzare le classi IFS per copiare un file da un indirizzario a un altro sull'iSeries.

"Esempio: utilizzo delle classi IFS per elencare il contenuto di un indirizzario" a pagina 501 mostra come utilizzare le classi IFS per elencare il contenuto di un indirizzario sull'iSeries.

Classe IFSFile:

La classe IFSFile rappresenta un oggetto nell'IFS iSeries.I metodi di IFSFile rappresentano operazioni che vengono effettuate sull'oggetto per intero. E' possibile utilizzare IFSFileInputStream, IFSFileOutputStream e IFSRandomAccessFile per leggere e scrivere sul file. La classe IFSFile consente al programma Java di eseguire queste operazioni:

- Determinare se l'oggetto esiste e se è un indirizzario o un file
- Determinare se il programma Java può leggere da o scrivere su un file
- Determinare la lunghezza di un file
- Determinare le autorizzazioni di un oggetto e impostare le autorizzazioni di un oggetto
- Creare un indirizzario
- Cancellare un file o un indirizzario
- Ridenominare un file o un indirizzario
- Richiamare o impostare l'ultima data di modifica di un file
- Elencare il contenuto di un indirizzario
- Elencare il contenuto di un indirizzario e salvare le informazioni attributo su una cache locale
- Determinare la quantità di spazio disponibile sul sistema
- Determinare il tipo dell'oggetto file quando si trova nel file system QSYS.LIB

E' possibile richiamare l'elenco di file in un indirizzario utilizzando sia il metodo list() che il metodo listFiles():

- Il metodo listFiles() memorizza nella cache le informazioni per ogni file sulla chiamata iniziale. Dopo aver chiamato listFiles(), l'utilizzo di altri metodi per richiamare i dettagli del file produce prestazioni migliori perché le informazioni vengono richiamate dalla cache. Ad esempio, la chiamata di isDirectory() ad un oggetto IFSFile restituito da listFiles() non richiede una chiamata al server.
- Il metodo list() invia al server una richiesta di informazioni per ogni singolo file; in questo modo il server viene rallentato e richiede più risorse.

Nota: l'utilizzo di listFiles() implica che le informazioni nella cache potrebbero diventare obsolete; quindi potrebbe essere necessario aggiornare i dati chiamando nuovamente listFiles().

Esempi

Gli esempi che seguono mostrano come utilizzare la classe IFSFile:

- “Esempio: creazione di un indirizzario” a pagina 494
- “Esempio: utilizzo delle eccezioni IFSFile per tenere traccia degli errori ” a pagina 495
- “Esempio: elenco dei file con un’estensione .txt” a pagina 496
- “Esempio: utilizzo del metodo IFSFile listFiles() per elencare il contenuto di un indirizzario” a pagina 496

Classe IFSJavaFile:

La classe IFSJavaFile rappresenta un file nell’IFS iSeries ed estende la classe java.io.File.IFSJavaFile consente di scrivere i file per l’interfaccia java.io.File per l’accesso agli IFS iSeries.

IFSJavaFile fa sì che interfacce portatili siano compatibili con java.io.File e utilizza gli errori e le eccezioni utilizzati da java.io.File. IFSJavaFile utilizza le funzioni di gestione della sicurezza da java.io.File, ma a differenza di java.io.File, IFSJavaFile utilizza continuamente le funzioni per la gestione della sicurezza.

E’ possibile utilizzare IFSJavaFile con IFSFileInputStream e IFSFileOutputStream. Non supporta java.io.FileInputStream e java.io.FileOutputStream.

IFSJavaFile si basa su IFSFile; tuttavia, la sua interfaccia è più simile a quella di java.io.File che a quella di IFSFile. IFSFile è un’alternativa alla classe IFSJavaFile.

E’ possibile richiamare l’elenco di file in un indirizzario utilizzando sia il metodo list() che il metodo listFiles():

- Il metodo listFiles() è più adatto perché richiama e memorizza nella cache le informazioni per ogni file al momento della chiamata iniziale. In seguito, le informazioni relative a ogni file vengono richiamate dalla cache.
- Il metodo list() invia una richiesta di informazioni per ogni singolo file; in questo modo il server viene rallentato e richiede più risorse.

Nota: l’utilizzo di listFiles() implica che le informazioni nella cache potrebbero diventare obsolete; di conseguenza potrebbe risultare necessario aggiornare nuovamente i dati.

Esempio: utilizzo di IFSJavaFile

Nota: leggere l’Esonero di responsabilità per gli esempi di codice per informazioni legali importanti.

Il seguente esempio mostra come utilizzare la classe IFSJavaFile:

```
// Gestire /Dir/File.txt sul flash di sistema.
AS400 as400 = new AS400("flash");
IFSJavaFile file = new IFSJavaFile(as400, "/Dir/File.txt");

// Determinare l'indirizzario principale del file.
String directory = file.getParent();

// Determinare il nome del file.
String name = file.getName();

// Determinare la dimensione del file.
long length = file.length();

// Determinare quando il file è stato modificato l'ultima volta.
Date date = new Date(file.lastModified());

// Cancellare il file.
```

```
  if (file.delete() == false)
{
     // Visualizzare il codice di errore.
  System.err.println("Unable to delete file.");
}

                            try
{
  IFSFileOutputStream os =
     new IFSFileOutputStream(file.getSystem(), file, IFSFileOutputStream.SHARE_ALL, false);
      byte[] data = new byte[256];
      int i = 0;
      for (; i < data.length; i++)
  {
         data[i] = (byte) i;
         os.write(data[i]);
  }
                 os.close();
}
                            catch (Exception e)
{
     System.err.println ("Exception: " + e.getMessage());
}
```

IFSFileInputStream:

La classe IFSFileInputStream rappresenta un flusso di immissione per la lettura dei dati da un file sul server.Così come nella classe IFSFile, in IFSFileInputStream esistono metodi in grado di duplicare i metodi in FileInputStream dal pacchetto java.io. In aggiunta a questi metodi, IFSFileInputStream dispone di metodi aggiuntivi specifici per i server iSeries. La classe IFSFileInputStream consente a un programma Java di eseguire queste operazioni:

- Aprire un file per la lettura.Il file deve esistere poiché questa classe non crea i file sul server. E' possibile utilizzare un programma di creazione che consenta di specificare i metodi di condivisione file.
- Determinare il numero di byte nel flusso.
- Leggere i byte dal flusso.
- Ignorare i byte nel flusso.
- Bloccare o sbloccare i byte nel flusso.
- Chiudere il file.

Così come in FileInputStream in java.io, questa classe consente al programma Java di leggere un flusso di byte dal file. Il programma Java legge i byte in modo sequenziale con la sola opzione aggiuntiva di ignorare i byte nel flusso.

In aggiunta ai metodi in FileInputStream, IFSFileInputStream fornisce al programma Java le seguenti opzioni:

- Bloccare e sbloccare i byte nel flusso. Per ulteriori informazioni, consultare IFSKey.
- Specificare un metodo di condivisione quando il file viene aperto. Per ulteriori informazioni, consultare metodi di condivisione.

Esempio: utilizzo di IFSFileInputStream

L'esempio che segue mostra come utilizzare la classe IFSFileInputStream.

```
                        // Creare un oggetto AS400.
AS400 sys = new AS400("mySystem.myCompany.com");

                                       // Aprire un oggetto file che
                                       // rappresenta il file.
IFSFileInputStream aFile = new IFSFileInputStream(sys,"/mydir1/mydir2/myfile");
```

```
                                        // Determinare il numero di byte nel
                                        // file.
        int available = aFile.available();

                                        // Assegnare un buffer per contenere i dati
        byte[] data = new byte[10240];

                                        // Leggere l'intero file 10K alla volta
        for (int i = 0; i < available; i += 10240)
    {
            aFile.read(data);
    }

                                        // Chiudere il file.
        aFile.close();
```

Classe IFSTextFileInputStream:

La classe IFSTextFileInputStream rappresenta un flusso di dati carattere letti da un file.I dati letti dall'oggetto IFSTextFileInputStream vengono forniti al programma Java in un oggetto Java String, in modo tale che siano sempre Unicode. Quando il file viene aperto, l'oggetto IFSTextFileInputStream determina il CCSID dei dati nel file. Se i dati vengono memorizzati in una codifica diversa da Unicode, l'oggetto IFSTextFileInputStream converte i dati dalla codifica del file in Unicode prima di fornire i dati al programma Java. Se i dati non possono essere convertiti, viene lanciato UnsupportedEncodingException.

L'esempio che segue mostra come utilizzare IFSTextFileInputStream:

```
                                        // Gestire /File sul sistema
                                        // mySystem.
        AS400 as400 = new AS400("mySystem");
        IFSTextFileInputStream file = new IFSTextFileInputStream(as400, "/File");

                                        // Leggere i primi quattro caratteri del
                                        // file.
        String s = file.read(4);

                    // Visualizzare i caratteri letti. Leggere,
                    // i primi quattro caratteri del
                    // file.
Se necessario, i dati vengono
                                        // convertiti in Unicode da
                                        // IFSTextFileInputStream.
        System.out.println(s);

                                        // Chiudere il file.
        file.close();
```

Classe IFSFileOutputStream:

La classe IFSFileOutputStream rappresenta un flusso di emissione per scrivere dati su un file sul server.Così come nella classe IFSFile, in IFSFileOutputStream esistono metodi in grado di duplicare i metodi in FileOutputStream dal pacchetto java.io. Inoltre IFSFileOutputStream dispone di metodi aggiuntivi specifici per il server. La classe IFSFileOutputStream consente al programma Java di eseguire queste operazioni:

- Aprire un file per la scrittura.Se il file esiste già, esso viene sostituito. Sono disponibili programmi di creazione che specificano i metodi di condivisione file e indicano se i contenuti di un file esistente sono stati aggiunti.
- Scrivere i byte sul flusso.
- Sincronizzare sul disco i byte scritti sul flusso.
- Bloccare o sbloccare i byte nel flusso.
- Chiudere il file.

Così come in FileOutputStream in java.io, questa classe consente al programma Java di scrivere in modo sequenziale un flusso di byte sul file.

In aggiunta ai metodi in FileOutputStream, IFSFileOutputStream fornisce al programma Java le seguenti opzioni:

- Bloccare e sbloccare i byte nel flusso. Per ulteriori informazioni, consultare IFSKey.
- Specificare un metodo di condivisione quando il file viene aperto. Per ulteriori informazioni, consultare metodi di condivisione.

Esempio: utilizzo di IFSFileOutputStream

Il seguente esempio mostra come utilizzare la classe IFSFileOutputStream:

```
                                        // Creare un oggetto AS400
AS400 sys = new AS400("mySystem.myCompany.com");

                                        // Aprire un oggetto file che
                                        // rappresenta il file.
IFSFileOutputStream aFile = new IFSFileOutputStream(sys,"/mydir1/mydir2/myfile");

                                        // Scrivere nel file
     byte i = 123;
     aFile.write(i);

                                        // Chiudere il file.
     aFile.close();
```

Classe IFSTextFileOutputStream:

La classe IFSTextFileOutputStream rappresenta un flusso di dati di caratteri scritti su un file.I dati forniti sull'oggetto IFSTextFileOutputStream si trovano nell'oggetto Java String in modo tale che l'immissione sia sempre Unicode. Tuttavia, l'oggetto IFSTextFileOutputStream può convertire i dati in un altro CCSID così come scritto nel file. La funzionalità predefinita è quella di scrivere caratteri Unicode nel file, ma il programma Java può impostare la destinazione CCSID prima che il file venga aperto. In questo caso, l'oggetto IFSTextFileOutputStream converte i caratteri da Unicode nel CCSID specifico prima di scriverli sul file. Se i dati non possono essere convertiti, viene lanciato UnsupportedEncodingException.

Esempio: utilizzo di IFSTextFileOutputStream

L'esempio che segue mostra come utilizzare IFSTextFileOutputStream:

```
                                        // Gestire /File sul sistema
                                        // mySystem.
     AS400 as400 = new AS400("mySystem");
     IFSTextFileOutputStream file = new IFSTextFileOutputStream(as400, "/File");

                                        // Scrivere una Stringa nel file.
                                        // Poiché non è stato specificato alcun CCSID
                                        // prima di scrivere nel file,
                                        // caratteri Unicode saranno
               // scritti nel file. Il file
                                        // sarà contrassegnato come se avesse dati
               // Unicode.
     file.write("Hello world");

                                        // Chiudere il file.
     file.close();
```

IFSRandomAccessFile:

La classe IFSRandomAccessFile rappresenta un file sul server per la lettura e la scrittura dei dati. Il programma Java può scrivere e leggere dati sia in modo sequenziale che casuale. Come in IFSFile, in

IFSRandomAccessFile esistono metodi in grado di duplicare i metodi in RandomAccessFile dal pacchetto java.io. In aggiunta a questi metodi, IFSRandomAccessFile dispone di metodi aggiuntivi specifici per i server iSeries. Attraverso IFSRandomAccessFile, un programma Java può eseguire queste operazioni:

- Aprire un file per la lettura, la scrittura o lettura/scrittura. Il programma Java può specificare facoltativamente il metodo di condivisione file e l'opzione di esistenza.
- Leggere dati sullo scostamento corrente dal file.
- Scrivere dati sullo scostamento corrente sul file.
- Richiamare o impostare lo scostamento corrente del file.
- Chiudere il file.

In aggiunta ai metodi in java.io RandomAccessFile, IFSRandomAccessFile fornisce al programma Java le opzioni seguenti:

- Sincronizzare sul disco i byte scritti.
- Bloccare o sbloccare i byte nel file.
- Bloccare e sbloccare i byte nel flusso. Per ulteriori informazioni, consultare IFSKey.
- Specificare un metodo di condivisione quando il file viene aperto. Per ulteriori informazioni, consultare metodi di condivisione.
- Specificare l'opzione di esistenza quando un file viene aperto. Il programma Java può scegliere una delle seguenti opzioni:
  - Aprire il file se esiste; creare il file se non esiste.
  - Sostituire il file se esiste; creare il file se non esiste.
  - Non riuscire ad aprire il file se quest'ultimo esiste; creare il file se non esiste.
  - Aprire il file se esiste; non riuscire ad eseguire tale operazione se il file non esiste.
  - Sostituire il file se esiste; non riuscire ad eseguire tale operazione se il file non esiste.

Esempio: utilizzo di IFSRandomAccessFile

L'esempio che segue mostra come utilizzare la classe IFSRandomAccessFile per scrivere quattro byte a intervalli 1K su un file.

```
                    // Creare un oggetto AS400.
AS400 sys = new AS400("mySystem.myCompany.com");

                                        // Aprire un oggetto file che rappresenta un
                                        // file.
IFSRandomAccessFile aFile = new IFSRandomAccessFile(sys,"/mydir1/myfile", "rw");

                                        // Stabilire i dati da scrivere.
     byte i = 123;

                                        // Scrivere nel file 10 volte ad intervalli
                                        // di 1K.
     for (int j=0; j<10; j++)
{
                                        // Spostare lo scostamento corrente.
           aFile.seek(j * 1024);

                  // Scrivere nel file. Lo scostamento
                                        // corrente avanza in base alla dimensione della
                                        // scrittura.
        aFile.write(i);
}

                                        // Chiudere il file.
     aFile.close();
```

IFSFileDialog:

La classe IFSFileDialog consente di consultare il file system e di selezionare un file. Questa classe utilizza la classe IFSFile per consultare l'elenco di indirizzari e di file nell'IFS sul server iSeries. I metodi sulla classe consentono ai programmi Java di impostare il testo sui pulsanti della finestra di dialogo per impostare i filtri. Tenere presente che è disponibile anche una classe IFSFileDialog basata su Swing 1.1.

E' possibile impostare i filtri con la classe FileFilter.Se si seleziona un file nella finestra di dialogo, il metodo getFileName() può essere utilizzato per richiamare il nome del file selezionato.Il metodo getAbsolutePath() può essere utilizzato per richiamare il percorso e il nome del file selezionato.

Esempio: utilizzo di IFSFileDialog

L'esempio che segue mostra come impostare una finestra di dialogo con due filtri e come impostare il testo sui pulsanti della finestra di dialogo.

```
                    // Creare un oggetto AS400.
AS400 sys = new AS400("mySystem.myCompany.com");

                                        // Creare un oggetto finestra di dialogo che imposta
                                        // il testo della barra del titolo della finestra
                                        // di dialogo ed il server su traverse.
     IFSFileDialog dialog = new IFSFileDialog(this, "Title Bar Text", sys);

                                        // Creare un elenco di filtri, quindi impostare
                    // i filtri nella finestra di dialogo. Il
                                        // primo filtro verrà utilizzato quando
                                        // la finestra di dialogo viene visualizzata per la prima volta.
     FileFilter[] filterList = {new FileFilter("All files (*.*)", "*.*"),
                                       new FileFilter("HTML files (*.HTML", "*.HTM")};

     dialog.setFileFilter(filterList, 0);

                                        // Impostare il testo sui pulsanti della
                                        // finestra di dialogo.
     dialog.setOkButtonText("Open");
     dialog.setCancelButtonText("Cancel");

                    // Visualizzare la finestra di dialogo. Se l'utente ha
                                       // selezionato un file premendo il pulsante
                                        // pulsante Apri, richiamare il file
                                        // selezionato dall'utente e visualizzarlo.
     if (dialog.showDialog() == IFSFileDialog.OK)
           System.out.println(dialog.getAbsolutePath());
```

Classe IFSKey:

Se il programma Java consente ad altri programmi di accedere ad un file nello stesso momento, il programma Java può bloccare i byte nel file per un dato periodo di tempo. Durante tale periodo, il programma utilizza in modo esclusivo la sezione del file in questione. Quando l'operazione di blocco riesce, la classe IFS restituisce un oggetto IFSKey.Tale oggetto viene fornito al metodo unlock() per individuare i byte da sbloccare. Quando il file viene chiuso, il sistema sblocca tutti i file ancora bloccati (il sistema esegue uno sblocco per ogni blocco non ancora disattivato dal programma).

Esempio: utilizzo di IFSKey

L'esempio che segue mostra come utilizzare la classe IFSKey.

```
                         // Creare un oggetto AS400.
AS400 sys = new AS400("mySystem.myCompany.com");

                    // Aprire un flusso di immissione. Tale
                                        // programma di creazione si apre con share_all
                                        // così altri programmi possono aprire questo
                                        // file.
```

```
          IFSFileInputStream aFile =
                                     new IFSFileInputStream(sys,"/mydir1/mydir2/myfile");

                                             // Bloccare i primi 1K di byte nel
                                             // file.
Ora, nessun'altra istanza è in grado di
                                             // leggere questi byte.
          IFSKey key = aFile.lock(1024);

                                             // Leggere il primo 1K del file.
          byte data[] = new byte[1024];
              aFile.read(data);

                                             // Sbloccare i byte del file.
          aFile.unlock(key);

                                             // Chiudere il file.
          aFile.close();
```

Metodi di condivisione file:

Il programma Java può specificare un metodo di condivisione quando un file viene aperto. Il programma consente ad altri programmi di aprire il file nello stesso momento o dispone di un accesso esclusivo al file.

L'esempio che segue mostra come specificare un metodo di condivisione file.

```
                       // Creare un oggetto AS400.
     AS400 sys = new AS400("mySystem.myCompany.com");

                                             // Aprire un oggetto file che
                                             // rappresenta il file.
Poiché questo
                                             // programma specifica share-none, tutti
                                             // gli altri tentativi aperti danno esito negativo fino
                                             // alla chiusura di questa istanza.
     IFSFileOutputStream aFile = new IFSFileOutputStream(sys,
                                                         "/mydir1/mydir2/myfile",
                                                         IFSFileOutputStream.SHARE_NONE,
                                                         false);

                       // Eseguire delle operazioni sul file.

                                             // Chiudere il file.
Ora, altre richieste
                                             // aperte hanno avuto esito positivo.
          aFile.close();
```

JavaApplicationCall

La classe JavaApplicationCall consente al client di utilizzare il server JVM per eseguire il programma Java che risiede nel server.

Dopo aver stabilito un collegamento al server dal client, la classe JavaApplicationCall consente di configurare quanto segue:

1. Impostare la variabile di ambiente CLASSPATH sul server con il metodo setClassPath()
2. Definire i parametri del programma con il metodo setParameters()
3. Eseguire il programma con run()
4. Inviare l'immissione dal client al programma Java. Il programma Java legge le immissioni tramite immissioni standard che vengono impostate con il metodo sendStandardInString().E' possibile reindirizzare le emissioni standard e gli errori standard dal programma Java al client attraverso getStandardOutString() e getStandardErrorString()

JavaApplicationCall è una classe che si richiama dal programma Java. Tuttavia, IBM Toolbox per Java fornisce programmi di utilità per richiamare i programmi Java che risiedono sul server. Questi programmi di utilità sono programmi Java completi che è possibile eseguire dalla propria stazione di lavoro. Per ulteriori informazioni, consultare la classe RunJavaApplication.

Esempio

L'esempio riportato nella documentazione di riferimento Javadoc di JavaApplicationCall mostra come eseguire un programma sul server (che emette ″Hello World!″) dal client:

JavaApplicationCall

Classi JDBC

JDBC(TM) è un'API (application programming interface) inclusa nella piattaforma Java che consente ai programmi Java di collegarsi ad una ampia gamma di database.

Per ulteriori informazioni su JDBC e sul supporto IBM Toolbox per Java per JDBC, inclusi i miglioramenti, le proprietà JDBC e i tipi SQL non supportati, consultare la seguente pagina:

"JDBC" a pagina 324

Interfacce supportate

La seguente tabella elenca le interfacce JDBC supportate e l'API necessaria per utilizzarle:

Interfaccia JDBC supportata	**API necessaria**
Blob fornisce l'accesso ai BLOB	JDBC 2.1 principale
CallableStatement esegue le procedure memorizzate SQL	JDK 1.1
Clob fornisce l'accesso ai CLOB	JDBC 2.1 principale
Connection rappresenta un collegamento ad un determinato database	JDK 1.1
ConnectionPool rappresenta un lotto di oggetti di collegamento.	JDBC 2.0 Optional Package
ConnectionPoolDataSource rappresenta una produzione per gli oggetti di lotto AS400JDBCPooledConnection	JDBC 2.0 Optional Package
DatabaseMetaData fornisce informazioni sull'intero database.	JDK 1.1
DataSource rappresenta una produzione per i collegamenti al database.	JDBC 2.0 Optional Package
Il programma di controllo crea il collegamento e restituisce informazioni sulla versione del programma di controllo.	JDK 1.1
ParameterMetaData consente di richiamare informazioni sui tipi e sulle proprietà dei parametri in un oggetto PreparedStatement	API di JDBC 3.0
PreparedStatement esegue istruzioni compilate SQL	JDK 1.1
ResultSet fornisce l'accesso alla tabella dei dati che vengono create dall'esecuzione di una interrogazione SQL o del metodo di catalogo DatabaseMetaData	JDK 1.1
ResultSetMetaData fornisce informazioni su uno specifico ResultSet	JDK 1.1

Interfaccia JDBC supportata	API necessaria
RowSet è una serie di righe connesse che contengono un ResultSet	JDBC 2.0 Optional Package
Savepoint fornisce un controllo particolareggiato nelle transazioni	API di JDBC 3.0
Istruzione esegue istruzioni SQL ed ottiene i risultati	JDK 1.1
XAConnection è un collegamento database che partecipa alle transazioni XA globali	JDBC 2.0 Optional Package
XAResource è un gestore risorse che si utilizza nelle transazioni XA	JDBC 2.0 Optional Package

Esempi

I seguenti esempi illustrano i modi di utilizzare l'unità di controllo JDBC IBM Toolbox per Java.

- "Esempio: utilizzo di JDBCPopulate per creare e popolare una tabella" a pagina 504
- "Esempio: utilizzo di JDBCQuery per interrogare una tabella" a pagina 506

Classe AS400JDBCBlob:

E' possibile utilizzare un oggetto AS400JDBCBlob per accedere ai BLOB (binary large object), come i file audio byte (.wav) o di immagine (.gif).

La differenza chiave tra la classe AS400JDBCBlob e la classe AS400JDBCBlobLocator è l'ubicazione in cui il blob viene memorizzato. Con la classe AS400JDBCBlob, il blob viene memorizzato nel database, che aumenta la dimensione del file database. La classe AS400JDBCBlobLocator memorizza un programma di ricerca (pensarlo come un puntatore) nel file database che punta all'ubicazione del blob.

Con la classe AS400JDBCBlob, è possibile utilizzare la proprietà della soglia del lob. Questa proprietà specifica la dimensione massima del LOB (in kilobyte) che è possibile richiamare come parte di una serie di risultati. I LOB che superano questa soglia sono richiamati in parti utilizzando comunicazioni supplementari con il server. Le soglie LOB più ampie riducono la frequenza delle comunicazioni con il server, ma scaricano più dati LOB, anche se queste non vengono utilizzate. Soglie del LOB più ridotte possono aumentare la frequenza delle comunicazioni con il server, ma scaricano i dati lob solo se necessari. Consultare Proprietà JDBC per informazioni su ulteriori proprietà disponibili.

Utilizzando la classe AS400JDBCBlob, è possibile effettuare le seguenti operazioni:

- Restituire l'intero blob come un flusso di byte non interpretati
- Restituire parte del contenuto del blob
- Restituire la lunghezza del blob
- Creare un flusso binario da scrivere nel blob
- Scrivere una schiera di byte sul blob
- Scrivere tutta o una parte della schiera di byte sul blob
- Troncare il blob

Esempi

I seguenti esempi, mostrano come utilizzare la classe AS400JDBCBlob per leggere o aggiornare un blob:

Esempio: utilizzo della classe AS400JDBCBlob per leggere da un blob

```
Blob blob = resultSet.getBlob(1);
     long length = blob.length();
     byte[] bytes = blob.getBytes(1, (int) length);
```

Esempio: utilizzo della classe AS400JDBCBlob per aggiornare un blob

```
ResultSet rs = statement.executeQuery ("SELECT BLOB FROM MYTABLE");
     rs.absolute(5);
     Blob blob = rs.getBlob(1);
               // Modificare i byte nel blob, a partire dal settimo byte
               // del blob
     blob.setBytes (7, new byte[] { (byte) 57, (byte) 58, (byte) 98});
               //Aggiornare il blob nella serie di risultati, modificando il blob a partire
               // dal settimo byte del blob (a base 1) e troncando il
               // blob alla fine dei byte aggiornati (il blob ora ha 9 byte).
     rs.updateBlob(1, blob);
     // Aggiornare il database con le modifiche. Questa operazione modificherà il blob
               // nel database a partire dal settimo byte del blob e
               // troncando alla fine dei byte aggiornati.
     rs.updateRow();
     rs.close();
```

Classe AS400JDBCBlobLocator

E' possibile utilizzare un oggetto AS400JDBCBlobLocator per accedere a oggetti binari ampi.

Utilizzando la classe AS400JDBCBlobLocator, è possibile effettuare le seguenti operazioni:

- Restituire l'intero blob come un flusso di byte non interpretati
- Restituire parte del contenuto del blob
- Restituire la lunghezza del blob
- Creare un flusso binario da scrivere nel blob
- Scrivere una schiera di byte sul blob
- Scrivere tutta o una parte della schiera di byte sul blob
- Troncare il blob

Interfaccia CallableStatement:

E' possibile utilizzare un oggetto CallableStatement per eseguire le procedure memorizzate SQL.La procedura memorizzata che viene richiamata deve essere già memorizzata nel database. CallableStatement non contiene la procedura memorizzata, esso richiama solo la procedura memorizzata.

Una procedura memorizzata può restituire uno o più oggetti ResultSet e può utilizzare parametri IN, OUT e INOUT. Utilizzare Connection.prepareCall() per creare nuovi oggetti CallableStatement.

L'oggetto CallableStatement consente di inoltrare più comandi SQL come un singolo gruppo ad un database tramite l'utilizzo del supporto batch. E' possibile ottenere prestazioni migliori utilizzando il supporto batch perché l'elaborazione di un gruppo di operazioni è, di solito, più veloce dell'elaborazione di una singola operazione alla volta. Per ulteriori informazioni sull'utilizzo del supporto batch, consultare Potenziamenti del supporto JDBC.

CallableStatement consente di richiamare e impostare i parametri e le colonne per nome, anche se l'utilizzo dell'indice delle colonne fornisce prestazioni migliori.

Esempio: utilizzo di CallableStatement

Il seguente esempio mostra come utilizzare l'interfaccia CallableStatement.

```
                              // Collegarsi al server.
     Connection c  = DriverManager.getConnection("jdbc:as400://mySystem");
                                          // Creare l'oggetto CallableStatement.
                         // AS400TreePane.
Precompila la chiamata
                                          // specificata in una procedura
                         // memorizzata. I punti interrogativi
                                          // indicano dove devono essere impostati
                                          // i parametri di immissione e dove possono
                                          // essere reperiti quelli di emissione.
                                          // I primi due parametri sono di immissione
                                          // mentre il terzo parametro è di
                                          // emissione.
     CallableStatement cs = c.prepareCall("CALL MYLIBRARY.ADD (?, ?, ?)");

                                          // Impostare i parametri di immissione.
          cs.setInt (1, 123);
          cs.setInt (2, 234);

                                          // Registrare il tipo di parametro di
                                          // emissione.
          cs.registerOutParameter (3, Types.INTEGER);

                                          // Eseguire la procedura memorizzata.
          cs.execute ();

                                          // Richiamare il valore del parametro di
                                          // emissione.
          int sum = cs.getInt (3);

                                          // Chiudere CallableStatement e
                                          // Connection.
          cs.close();
          c.close();
```

Classe AS400JDBCClob:

E' possibile utilizzare un oggetto AS400JDBCClob per accedere ai CLOB (character large object), come ad esempio documenti estesi.

La differenza fondamentale tra la classe AS400JDBCClob e la classe AS400JDBCClobLocator è costituita dall'ubicazione in cui viene memorizzato il blob. Con la classe AS400JDBCClob, il clob viene memorizzato nel database, che estende la dimensione del file del database. La classe AS400JDBCClobLocator memorizza un programma di ubicazione (da considerarsi come se fosse un puntatore) nel file del database che punta l'ubicazione del clob.

Con la classe AS400JDBCClob, può essere utilizzata la proprietà soglia del lob. Questa proprietà specifica la dimensione massima del LOB (in kilobyte) che è possibile richiamare come parte di una serie di risultati. I LOB che superano questa soglia sono richiamati in parti utilizzando comunicazioni supplementari con il server. Le soglie LOB più ampie riducono la frequenza delle comunicazioni con il server, ma scaricano più dati LOB, anche se queste non vengono utilizzate. Soglie del LOB più ridotte possono aumentare la frequenza delle comunicazioni con il server, ma scaricano i dati lob solo se necessari. Consultare "Proprietà JDBC di IBM Toolbox per Java" a pagina 328 per informazioni su altre proprietà disponibili.

Utilizzando la classe AS400JDBCClob, è possibile effettuare le seguenti operazioni:

- Restituire l'intero clob come un flusso di caratteri ASCII
- Restituire il contenuto del clob come un flusso di caratteri
- Restituire una parte del contenuto del clob
- Restituire la lunghezza del clob

- Creare un flusso di caratteri Unicode o un flusso di caratteri ASCII per scrivere nel clob
- Scrivere una stringa nel clob
- Troncare il clob

Esempi

I seguenti esempi mostrano come utilizzare la classe AS400JDBCClob per leggere e aggiornare un clob:

Esempio: utilizzo della classe AS400JDBCClob per leggere da un clob

```
Clob clob = rs.getClob(1);
int length = clob.length();
String s = clob.getSubString(1, (int) length);
```

Esempio: utilizzo della classe AS400JDBCClob per aggiornare un clob

```
ResultSet rs = statement.executeQuery ("SELECT CLOB FROM MYTABLE");
     rs.absolute(4);
     Clob clob = rs.getClob(1);

               // Modificare i caratteri nel clob, iniziando dal terzo carattere
               // del clob
clob.setString (3, "Small");

               // Aggiornare il clob nel resultset, iniziando dal terzo carattere
               // del clob e troncando il clob alla fine della stringa di aggiornamento
               // (ora il clob ha 7 caratteri).
     rs.updateClob(1, clob);

     // Aggiornare il database con il clob aggiornato. Questa operazione modificherà il
               // clob nel database iniziando dal terzo carattere del clob
               // e troncando alla fine della stringa di aggiornamento.
     rs.updateRow();
     rs.close();
```

Classe AS400JDBCClobLocator

E' possibile utilizzare un oggetto AS400JDBCClobLocator per accedere ai clob.

Utilizzando la classe AS400JDBCClobLocator, è possibile effettuare le seguenti operazioni:
- Restituire l'intero clob come un flusso di caratteri ASCII
- Restituire l'intero clob come un flusso di caratteri
- Restituire una parte del contenuto del clob
- Restituire la lunghezza del clob
- Creare un flusso di caratteri Unicode o un flusso di caratteri ASCII per scrivere nel clob
- Scrivere una stringa nel clob
- Troncare il clob

Classe AS400JDBCConnection:

La classe AS400JDBCConnection fornisce un collegamento JDBC ad uno specifico database UDB DB2 per iSeries. Utilizzare DriverManager.getConnection() per creare nuovi oggetti AS400JDBCConnection. Per ulteriori informazioni, consultare "Registrazione dell'unità di controllo JDBC" a pagina 71.

Ci sono molte proprietà facoltative che possono essere specificate quando si crea il collegamento. Le proprietà possono essere specificate come parte dell'URL o nell'oggetto java.util.Properties. Consultare "Proprietà JDBC di IBM Toolbox per Java" a pagina 328 per un elenco completo di proprietà supportate dalla classe AS400JDBCDriver.

Nota: un collegamento può contenere al massimo 9999 istruzioni aperte.

AS400JDBCConnection include un supporto per i punti di salvataggio, conservabilità al livello di istruzioni e supporto limitato per la restituzione delle chiavi autogenerate. Per ulteriori informazioni su questi e altri aggiornamenti, consultare “Aggiornamenti del supporto JDBC di IBM Toolbox per Java” a pagina 325.

Per utilizzare i ticket Kerberos, impostare solo il nome del sistema (e non la parola d’ordine) nell’oggetto URL di JDBC. L’identità dell’utente viene richiamata tramite la framework JGSS (Java Generic Security Services), in modo che non sia necessario specificare anche un utente nell’URL di JDBC. E’ possibile impostare solo un mezzo di autenticazione alla volta in un oggetto AS400JDBCConnection. L’impostazione della parola d’ordine eliminerà il contenuto di ogni ticket Kerberos o token del profilo. Per ulteriori informazioni, consultare la pagina “Classe AS400” a pagina 26 e J2SDK, Documentazione di sicurezza v1.4  .

Utilizzando la classe AS400JDBCConnection, è possibile effettuare le seguenti operazioni:

- Creare un’istruzione (oggetti Statement, PreparedStatement o CallableStatement)
- Creare una istruzione che disponga di un tipo serie di risultati e una convergenza specifici (Oggetti Statement, PreparedStatement o CallableStatement)
- Eseguire il commit e il Rollback delle modifiche al database e rilasciare i vincoli sul database attualmente congelati
- Chiudere il collegamento, chiudendo immediatamente le risorse del server invece di attendere che siano automaticamente rilasciate
- Impostare la conservabilità e richiamare la conservabilità per il collegamento
- Impostare l’isolamento della transazione e richiamare l’isolamento della transazione per il collegamento
- Richiamare i meta dati per il collegamento
- Impostare la sincronizzazione automatica su attivo o disattivo
- Richiamare l’identificativo del lavoro del lavoro server host che corrisponde al collegamento

Se si sta utilizzando JDBC 3.0 e si sta effettuando il collegamento al server su cui è in esecuzione V5R2 di OS/400 o versione successiva, è possibile utilizzare AS400JDBCConnection per effettuare le seguenti operazioni:

- Creare un’istruzione con una specifica conservabilità della serie dei risultati (oggetto Statement, PreparedStatement, o CallableStatement)
- Creare un’istruzione preparata che restituisca ogni chiave autogenerata (quando getGeneratedKeys() viene richiamato sull’oggetto Statement)
- Utilizzare punti di salvataggio, che offrono un controllo più approfondito sulle transazioni:
  - Impostare punti di salvataggio
  - Eseguire il rollback dei punti di salvataggio
  - Rilasciare punti di salvataggio

AS400JDBCConnectionPool:

La classe AS400JDBCConnectionPool rappresenta un lotto di oggetti AS400JDBCConnection disponibili per essere utilizzati da un programma Java come parte del supporto di IBM Toolbox per Java per l’API di JDBC 2.0 Optional Package.

E’ possibile utilizzare un AS400JDBCConnectionPoolDataSource per specificare le proprietà per i collegamenti che sono creati nel lotto, come nel seguente esempio.

Non è possibile modificare il sorgente dei dati del lotto di collegamenti dopo aver richiesto un collegamento e il lotto è in uso. Per ripristinare il sorgente dei dati del lotto di collegamenti, richiamare prima close() nel lotto.

Ripristinare i collegamenti ad un AS400JDBCConnectionPool tramite l'utilizzo del comando close() sull'oggetto AS400JDBCConnection.

Nota: quando i collegamenti non vengono restituiti al lotto, il lotto di collegamenti continua a crescere di dimensioni e i collegamenti non vengono riutilizzati.

Impostare le proprietà nel lotto utilizzando i metodi ereditati da ConnectionPool.Alcune proprietà che possono essere impostate includono:

- numero massimo di collegamenti consentiti nel lotto
- durata massima di un collegamento
- tempo massimo di inattività di un collegamento

E' possibile anche registrare gli oggetti AS400JDBCConnectionPoolDataSource tramite il tecnico di manutenzione JNDI (Java Naming and Directory Interface)(TM). Per ulteriori informazioni sui tecnici di manutenzione JNDI, consultare Collegamenti di riferimento IBM Toolbox per Java.

Esempio: utilizzo del lotto di collegamenti

Il seguente esempio richiama il sorgente dati del lotto di collegamenti da JNDI e lo utilizza per creare un lotto di collegamenti con 10 collegamenti:

```
          // Ottenere un oggetto AS400JDBCConnectionPoolDataSource da JNDI
          // (presuppone che l'ambiente JNDI sia impostato).
Context context = new InitialContext(environment);
AS400JDBCConnectionPoolDataSource datasource =
(AS400JDBCConnectionPoolDataSource)context.lookup("jdbc/myDatabase");

          // Creare un oggetto AS400JDBCConnectionPool.
AS400JDBCConnectionPool pool = new AS400JDBCConnectionPool(datasource);

          // Aggiunge 10 collegamenti al lotto che può essere utilizzato dalla
          // applicazione (crea i collegamenti fisici al database in base
          // all'origine dati).
pool.fill(10);

          // Richiamare un handle in un collegamento database del lotto.
Connection connection = pool.getConnection();

... Eseguire diverse interrogazioni/aggiornamenti sul database.

          // Chiudere l'handle di collegamento per restituirlo al lotto.
               connection.close ();

... L'applicazione gestisce alcuni collegamento dal lotto.

// Chiudere il lotto per rilasciare tutte le risorse.
pool.close();
```

Interfaccia DatabaseMetaData:

E' possibile utilizzare un oggetto DatabaseMetaData per ottenere informazioni sull'intero database così come sulle informazioni su catalogo.

Il seguente esempio mostra come ripristinare un elenco di tabelle, che è una funzione del catalogo:

```
                          // Collegarsi al server.
Connection c  = DriverManager.getConnection("jdbc:as400://mySystem");
                // Richiamare i meta dati database dal collegamento.
     DatabaseMetaData dbMeta = c.getMetaData();

                // Richiamare un elenco di tabelle corrispondenti ai seguenti criteri.
     String catalog = "myCatalog";
     String schema  = "mySchema";
     String table   = "myTable%"; // % indica il modello di ricerca
     String types[]  = {"TABLE", "VIEW", "SYSTEM TABLE"};
     ResultSet rs = dbMeta.getTables(catalog, schema, table, types);

                // Iterare ResultSet per richiamare i valori.

                                       // Chiudere il collegamento.
c.close();
```

Classe AS400JDBCDataSource:

La classe AS400JDBCDataSource rappresenta una produzione per i collegamenti database di iSeries. La classe AS400JDBCConnectionPoolDataSource rappresenta una produzione per gli oggetti AS400JDBCPooledConnection.

E' possibile registrare ogni tipo di oggetto sorgente dati tramite un tecnico di manutenzione JNDI. Per ulteriori informazioni sui tecnici di manutenzione JNDI, consultare Collegamenti di riferimento IBM Toolbox per Java.

Esempi

I seguenti esempi mostrano i modi per creare ed utilizzare gli oggetti AS400JDBCDataSourcets. Gli ultimi due esempi mostrano come registrare un oggetto AS400JDBCDataSource con JNDI e, quindi, utilizzare gli oggetti restituiti da JNDI per ottenere un collegamento database. Notare che anche quando vengono utilizzati diversi tecnici di manutenzione JNDI, il codice è molto simile.

Esempio: creazione di un oggetto AS400JDBCDataSource

Il seguente esempio mostra come creare un oggetto AS400JDBCDataSource e come collegarlo al database:

```
// Creare un'origine dati per stabilire un collegamento.
AS400JDBCDataSource datasource = new AS400JDBCDataSource("myAS400");
datasource.setUser("myUser");
datasource.setPassword("MYPWD");

// Creare un collegamento database ad iSeries.
Connection connection = datasource.getConnection();
```

Esempio: creazione di un oggetto AS400JDBCConnectionPoolDataSource che può essere utilizzato per memorizzare in cache i collegamenti JDBC

Il seguente esempio mostra come è possibile utilizzare AS400JDBCConnectionPoolDataSource per memorizzare in cache i collegamenti JDBC.

```
// Creare un'origine dati per stabilire un collegamento.
AS400JDBCConnectionPoolDataSource dataSource = new AS400JDBCConnectionPoolDataSource("myAS400");
datasource.setUser("myUser");
datasource.setPassword("MYPWD");

// Richiamare PooledConnection.
PooledConnection pooledConnection = datasource.getPooledConnection();
```

Esempio: utilizzo delle classi del tecnico di manutenzione JNDI per memorizzare un oggetto AS400JDBCDataSource

Il seguente esempio mostra come è possibile utilizzare le classi del tecnico di manutenzione JNDI per memorizzare un oggetto DataSource direttamente sull'IFS sul server:

```
// Creare un'origine dati nel database iSeries.
AS400JDBCDataSource dataSource = new AS400JDBCDataSource();
dataSource.setServerName("myAS400");
dataSource.setDatabaseName("myAS400 Database");

// Registrare l'origine dati con Java Naming and Directory Interface (JNDI).
Hashtable env = new Hashtable();
env.put(Context.INITIAL_CONTEXT_FACTORY, "com.sun.jndi.fscontext.RefFSContextFactory");
Context context = new InitialContext(env);
context.bind("jdbc/customer", dataSource);

// Restituire un oggetto AS400JDBCDataSource da JNDI e richiamare un collegamento.
AS400JDBCDataSource datasource = (AS400JDBCDataSource) context.lookup("jdbc/customer");
Connection connection = datasource.getConnection("myUser", "MYPWD");
```

Esempio: utilizzo degli oggetti AS400JDBCDataSource e delle classi IBM SecureWay Directory con un server indirizzario LDAP (Lightweight Directory Access Protocol)

I seguenti esempi mostrano è possibile utilizzare le classi dell'indirizzario SecureWay di IBM per memorizzare un oggetto nel server dell'indirizzario LDAP:

```
// Creare un'origine dati nel database iSeries.
AS400JDBCDataSource dataSource = new AS400JDBCDataSource();
dataSource.setServerName("myAS400");
dataSource.setDatabaseName("myAS400 Database");

// Registrare l'origine dati con Java Naming and Directory Interface (JNDI).
Hashtable env = new Hashtable();
env.put(Context.INITIAL_CONTEXT_FACTORY, "com.ibm.jndi.LDAPCtxFactory");
Context context = new InitialContext(env);
context.bind("cn=myDatasource, cn=myUsers, ou=myLocation,o=myCompany,c=myCountryRegion",
             dataSource);

// Restituire un oggetto AS400JDBCDataSource da JNDI e richiamare un collegamento.
AS400JDBCDataSource datasource = (AS400JDBCDataSource) context.lookup(
   "cn=myDatasource, cn=myUsers, ou=myLocation,o=myCompany,c=myCountryRegion");
Connection connection = datasource.getConnection("myUser", "MYPWD");
```

Registrazione dell'unità di controllo JDBC:

Prima di utilizzare JDBC per accedere ai dati in un file database del server, è necessario registrare l'unità di controllo JDBC per il programma su licenza IBM Toolbox per Java con DriverManager.E' possibile registrare il programma di controllo utilizzando una proprietà di sistema Java o registrando il programma di controllo con il programma Java:

- Registrazione utilizzando una proprietà di sistema

  Ogni macchina virtuale ha il suo metodo di impostazione delle proprietà di sistema. Ad esempio, il comando Java dal JDK utilizza l'opzione -D per impostare le proprietà di sistema. Per impostare il programma di controllo utilizzando le proprietà di sistema, specificare quanto segue:

  ```
  "-Djdbc.drivers=com.ibm.as400.access.AS400JDBCDriver"
  ```

- Effettuare la registrazione utilizzando il programma Java

  Per caricare l'unità di controllo JDBC di IBM Toolbox per Java, aggiungere quanto segue al programma Java prima della prima chiamata a JDBC:

  ```
  Class.forName("com.ibm.as400.access.AS400JDBCDriver");
  ```

  L'unità di controllo JDBC di IBM Toolbox per Java si registra durante il caricamento; questa è la modalità preferenziale di registrazione dell'unità di controllo. Inoltre, è possibile registrare esplicitamente l'unità di controllo JDBC di IBM Toolbox utilizzando quanto segue:

  ```
  java.sql.DriverManager.registerDriver(new com.ibm.as400.access.AS400JDBCDriver());
  ```

L'unità di controllo JDBC di IBM Toolbox per Java non richiede un oggetto AS400 come parametro di immissione come accade per le altre classi IBM Toolbox per Java che richiamano dati da un server. Tuttavia, un oggetto AS400 è utilizzato internamente per gestire l'utente predefinito e la memorizzazione in cache della parola d'ordine. Quando viene effettuato il primo collegamento al server, all'utente possono essere richiesti l'ID e la parola d'ordine. L'utente ha l'opzione di salvare l'ID utente come ID dell'utente predefinito ed aggiungere la parola d'ordine nella cache delle parole d'ordine. Come nelle altre funzioni di IBM Toolbox per Java, se l'ID utente e la parola d'ordine sono forniti dal programma Java, l'utente predefinito non è impostato e la parola d'ordine non viene memorizzata nella cache. Consultare "Gestione dei collegamenti" a pagina 449 per informazioni sulla gestione dei collegamenti.

Utilizzo dell'unità di controllo JDBC per il collegamento a un database sul server

E' possibile utilizzare il metodo DriverManager.getConnection() per collegarsi al database del server. DriverManager.getConnection() richiede una stringa URL (Uniform Resource Locator) come argomento. Il gestore dell'unità di controllo JDBC cerca di individuare un programma di controllo in grado di collegarsi al database rappresentato dall'URL. Durante l'esecuzione del programma di controllo IBM Toolbox per Java, utilizzare la seguente sintassi per l'URL:

```
"jdbc:as400://systemName/defaultSchema;listOfProperties"
```

Nota: sia systemName che defaultSchema possono essere omessi dall'URL.

Per utilizzare i ticket Kerberos, impostare solo il nome del sistema (e non la parola d'ordine) nell'oggetto URL di JDBC. L'identità dell'utente viene richiamata tramite la framework JGSS (Java Generic Security Services), in modo che non sia necessario specificare anche un utente nell'URL di JDBC. E' possibile impostare solo un mezzo di autenticazione alla volta in un oggetto AS400JDBCConnection. L'impostazione della parola d'ordine eliminerà il contenuto di ogni ticket Kerberos o token del profilo. Per ulteriori informazioni, consultare la pagina "Classe AS400" a pagina 26 e J2SDK, Documentazione di sicurezza v1.4 .

Esempi: utilizzo dell'unità di controllo JDB per collegarsi al server

Esempio: utilizzo di una URL in cui non è specificato un nome di sistema

In questo esempio, all'utente verrà richiesto di immettere il nome del sistema con il quale si desidera collegarsi.

```
                    // Stabilire un collegamento a un sistema non denominato.
                    // All'utente viene richiesto di immettere il nome di sistema.
Connection c = DriverManager.getConnection("jdbc:as400:");
```

Esempio: collegamento al database del server; nessuno schema predefinito o proprietà specificati

```
                    // Stabilire un collegamento al sistema 'mySystem'. Non
                    // vengono specificati schemi o proprietà
                    // predefiniti.
Connection c  = DriverManager.getConnection("jdbc:as400://mySystem");
```

Esempio: collegamento al database del server; schema predefinito specificato

```
                    // Stabilire un collegamento al sistema 'mySys2'. Viene
                    // specificato lo schema 'myschema'
                    // predefinito.
Connection c2 = DriverManager.getConnection("jdbc:as400://mySys2/mySchema");
```

Esempio: collegamento al database del server ed utilizzo di java.util.Properties per specificare le proprietà

Il programma Java può specificare una serie di proprietà JDBC utilizzando l'interfaccia java.util.Properties o specificando le proprietà come parte dell'URL. Consultare "Proprietà JDBC di IBM Toolbox per Java" a pagina 328 per un elenco di proprietà supportate.

Ad esempio, per specificare le proprietà utilizzando l'interfaccia Proprietà, utilizzare il seguente codice come esempio:

```
                    // Creare un oggetto proprietà.
Properties p = new Properties();

                    // Impostare le proprietà del
                    // il collegamento.
p.put("naming", "sql");
p.put("errors", "full");

                    // Collegarsi utilizzando l'oggetto
                    // AS400TreePane.
Connection c = DriverManager.getConnection("jdbc:as400://mySystem",p);
```

Esempio: collegamento al database del server e utilizzare una URL per specificare le proprietà

```
                    // Stabilire un collegamento utilizzando le proprietà. Le
                    // proprietà vengono impostate sull'URL
                    // invece che tramite un oggetto
                    // AS400TreePane.
Connection c = DriverManager.getConnection(
                     "jdbc:as400://mySystem;naming=sql;errors=full");
```

Esempio: collegamento al database del server e specificare l'ID utente e la parola d'ordine

```
                    // Collegarsi utilizzando le proprietà
                    // sull'URL e specificando un ID utente
                    // e una parola d'ordine
Connection c = DriverManager.getConnection(
                     "jdbc:as400://mySystem;naming=sql;errors=full",
                     "auser",
                     "apassword");
```

Esempio: scollegamento dal database Per scollegarsi dal server, utilizzare il metodo close() nell'oggetto Connecting. Utilizzare la seguente istruzione per chiudere un collegamento creato nel precedente esempio:

```
c.close();
```

Classe AS400JDBCParameterMetaData:

La classe AS400JDBCParameterMetaData consente al programma di richiamare informazioni sulle proprietà dei parametri negli oggetti PreparedStatement e CallableStatement.

AS400JDBCParameterMetaData fornisce i metodi che consentono di svolgere le seguenti operazioni:

- Richiamare il nome classe del parametro
- Richiamare il numero dei parametri in PreparedStatement
- Richiamare il tipo SQL del parametro
- Richiamare il nome del tipo specifico per il database relativo al parametro
- Richiamare la precisione o la scala del parametro

Esempio: utilizzo di AS400JDBCParameterMetaData

Il seguente esempio mostra un modo di utilizzare AS400JDBCParameterMetaData per richiamare i parametri da un oggetto PreparedStatement creato dinamicamente:

```
// Richiamare un collegamento dall'unità di controllo.
     Class.forName("com.ibm.as400.access.AS400JDBCDriver");
     Connection connection =
   DriverManager.getConnection("jdbc:as400://myAS400", "myUserId", "myPassword");

     // Creare un oggetto PreparedStatement.
     PreparedStatement ps =
   connection.prepareStatement("SELECT STUDENTS FROM STUDENTTABLE WHERE STUDENT_ID= ?");

     // Impostare un ID studente nel parametro 1.
     ps.setInt(1, 123456);

     // Reperire i meta dati del parametro per l'istruzione preparata.
     ParameterMetaData pMetaData = ps.getParameterMetaData();

     // Reperire il numero di parametro nell'istruzione preparata.
     // Restituisce 1.
     int parameterCount = pMetaData.getParameterCount();

     // Individuare il nome tipo parametro del parametro 1.
     // Restituisce INTEGER.
     String getParameterTypeName = pMetaData.getParameterTypeName(1);
```

Interfaccia PreparedStatement:

E' possibile utilizzare un oggetto PreparedStatement quando si ha intenzione di eseguire molte volte un'istruzione SQL.Un'istruzione SQL può essere precompilata. Un'istruzione ″prepared″ è un'istruzione SQL che è stata precompilata. Questo approccio è più efficiente rispetto all'esecuzione ripetuta della stessa istruzione utilizzando un oggetto Statement, che compila l'istruzione ogni volta che viene eseguita. Inoltre, l'istruzione SQL contenuta in un oggetto PreparedStatemen può avere uno o più parametri IN. Utilizzare Connection.prepareStatement() per creare oggetti PreparedStatement.

L'oggetto PreparedStatement consente di inoltrare più comandi SQL ad un database come singolo gruppo tramite l'utilizzo del supporto batch. E' possibile ottenere prestazioni migliori utilizzando il supporto batch in quanto l'elaborazione di un gruppo di operazioni risulta solitamente più veloce rispetto all'elaborazione di un gruppo alla volta. Per ulteriori informazioni sull'utilizzo del supporto batch, consultare Potenziamenti del supporto JDBC.

Esempio: utilizzo di PreparedStatement

Il seguente esempio mostra come utilizzare l'interfaccia PreparedStatement.

```
                              // Collegarsi al server.
     Connection c  = DriverManager.getConnection("jdbc:as400://mySystem");
                                        // Creare l'oggetto PreparedStatement.
                       // AS400TreePane.
Precompila
                       // l'istruzione SQL specificata. I punto interrogativi,
                                        // indicano dove devono essere impostati
                                        // i parametri prima che venga eseguita
                                        // l'istruzione.
     PreparedStatement ps = c.prepareStatement("INSERT INTO MYLIBRARY.MYTABLE (NAME, ID) VALUES (?, ?)");

                                        // Impostare i parametri ed eseguire
                                        // l'istruzione.
          ps.setString(1, "JOSH");
          ps.setInt(2, 789);
          ps.executeUpdate();

                                        // Impostare i parametri ed eseguire
                                        // di nuovo l'istruzione.
          ps.setString(1, "DAVE");
          ps.setInt(2, 456);
          ps.executeUpdate();
```

```
                                        // Chiudere PreparedStatement e
                                        // Connection.
        ps.close();
    c.close();
```

Classe ResultSet:

E' possibile utilizzare un oggetto ResultSet per accedere a una tabella di dati generati dall'esecuzione di una interrogazione.Le righe della tabella sono richiamate in sequenza. All'interno di una riga, è possibile accedere ai valori della colonna in qualsiasi ordine.

I dati memorizzati in ResultSet vengono richiamati utilizzando i vari metodi get, a seconda del tipo di dati richiamati.Il metodo next() è utilizzato per spostarsi alla riga successiva.

ResultSet consente di richiamare e aggiornare le colonne per nome, sebbene l'utilizzo dei risultati dell'indice delle colonne migliora le prestazioni.

Movimento cursore

Un cursore, che è un puntatore interno, viene utilizzato da una serie risultati per indicare la riga nella serie risultati alla quale accede il programma Java.

Le prestazioni del metodo getRow() sono state potenziate. Prima di V5R2, l'utilizzo di ResultSet.last(), ResultSet.afterLast() e ResultSet.absolute() con un valore negativo rendeva non disponibile il numero di riga corrente. Le precedenti restrizioni vengono eliminate, ciò rende il metodo getRow() pienamente funzionale.

Le specifiche JDBC 2.0 e le successive specifiche JDBC forniscono ulteriori metodi per accedere a determinate posizioni all'interno di un database:

Posizioni del cursore scorrevole	
absolute	isFirst
afterLast	isLast
beforeFirst	last
first	moveToCurrentRow
getRow	moveToInsertRow
isAfterLast	previous
isBeforeFirst	relative

Capacità di scorrimento

Se una serie risultati viene creata eseguendo un'istruzione, è possibile spostarsi (scorrere) all'indietro (dall'ultimo al primo) o in avanti (dal primo all'ultimo) attraverso le righe in una tabella.

Una serie risultati che supporta questo spostamento viene denominata serie risultati scorrevole. Le serie di risultati scorrevoli supportano anche il posizionamento assoluto. Il posizionamento assoluto consente di spostarsi direttamente alla riga specificando la sua posizione nella serie di risultati.

Con le specifiche JDBC 2.0 e le successive specifiche JDBC, si hanno due ulteriori capacità di scorrimento disponibili da utilizzare mentre si lavora con la classe ResultSet: le serie risultati insensibile e sensibile allo scorrimento.

Una serie di risultati insensibile allo scorrimento generalmente non è sensibile alla modifiche effettuate quando è aperta, mentre la serie di risultati sensibile allo scorrimento è sensibile alle modifiche.

Nota: IBM iSeries Server consenete solo un accesso di sola lettura per i cursori non sensibili di scorrimento. IBM Toolbox per Java supporta un cursore non sensibile di scorrimento se la combinazione della serie di risultati è di sola lettura. Se il tipo di serie di risultati viene specificato come non sensibile e la combinazione viene specificata come aggiornabile, il tipo serie di risultati viene modificato in sensibile ed emette un messaggio di avvertenza.

Serie di risultati aggiornabili

Nell'applicazione, è possibile utilizzare le serie di risultati che utilizzano la combinazione di sola lettura (non è possibile aggiornare i dati) o la combinazione aggiornabile (consente aggiornamenti ai dati e può utilizzare blocchi di scrittura del database per controllare l'accesso alle stesse voci di dati da parte di differenti transazioni). In una serie di risultati aggiornabile, è possibile aggiornare, inserire e cancellare le righe. Numerosi metodi di aggiornamento sono disponibili, ad esempio:

- Aggiornare il flusso ASCII
- Aggiornare il Big Decimal
- Aggiornare il flusso binario

Consultare Riepilogo metodi per un elenco completo di metodi di aggiornamento disponibili tramite l'interfaccia ResultSet.

Esempio: serie di risultati aggiornabili

Il seguente esempio mostra come utilizzare una serie di risultati che consente aggiornamenti ai dati (combinazione aggiornamento) e modifiche alla serie di risultati quando la serie dei risultati è aperta (sensibile allo scorrimento).

```
                                // Collegarsi al server.
        Connection c  = DriverManager.getConnection("jdbc:as400://mySystem");

                                // Creare un oggetto Statement.
Impostare la combinazione della serie di risultati su
                                // aggiornabile.
     Statement s = c.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE, ResultSet.CONCUR_UPDATABLE);

                      // Eseguire un'interrogazione. Il risultato viene inserito
                                // in un oggetto ResultSet.
     ResultSet rs = s.executeQuery ("SELECT NAME,ID FROM MYLIBRARY.MYTABLE FOR UPDATE");

                                // Iterare le righe del ResultSet.
                                        // Mentre la riga viene letta, verrà aggiornata con
                                            // un nuovo ID.
          int newId = 0;
          while (rs.next ())
     {

                                    // Richiamare i valori dal ResultSet.
                                    // Il primo valore è una stringa e
                                    // il secondo è un valore intero.
         String name = rs.getString("NAME");
         int id = rs.getInt("ID");

         System.out.println("Name = " + name);
         System.out.println("Old id = " + id);

                                    // Aggiornare l'id con un nuovo valore intero.
         rs.updateInt("ID", ++newId);

                                  // Inviare gli aggiornamenti al server.
                 rs.updateRow ();

         System.out.println("New id = " + newId);
     }
```

```
                                        // Chiudere Statement e
                                        // Connection.
    s.close();
    c.close();
```

ResultSetMetaData

L'interfaccia ResultSetMetaData determina i tipi e le proprietà delle colonne in un ResultSet.

Durante il collegamento al server su cui è in esecuzione OS/400 V5R2 o versione successiva, l'utilizzo della proprietà extended metadata consente di aumentare la precisione dei seguenti metodi ResultSetMetaData:

- getColumnLabel(int)
- isReadOnly(int)
- isSearchable(int)
- isWriteable(int)

Inoltre, impostare questa proprietà su true abilita il supporto per il metodo ResultSetMetaData.getSchemaName(int). Tenere presente che l'utilizzo della proprietà extended metadata potrebbe influire negativamente sulle prestazioni poiché richiama più informazioni dal server.

Classe AS400JDBCRowSet:

La classe AS400JDBCRowSet rappresenta un rowset collegato che incorpora una serie di risultati JDBC.I metodi su AS400JDBCRowSet sono molto simili a quelli diAS400JDBCResultSet.Il collegamento database è mantenuto durante l'utilizzo.

E' possibile utilizzare un'istanza di AS400JDBCDataSource o AS400JDBCConnectionPoolDataSource per creare un collegamento al database che si desidera utilizzare per accedere ai dati per AS400JDBCRowSet.

Esempi

I seguenti esempi mostrano come è possibile utilizzare la classe AS400JDBCRowSet.

Esempio: creazione, popolamento e aggiornamento di un oggetto AS400JDBCRowSet

```
DriverManager.registerDriver (new AS400JDBCDriver ());
// Stabilire un collegamento utilizzando un URL.
AS400JDBCRowSet rowset = new AS400JDBCRowSet("jdbc:as400://mySystem","myUser", "myPassword");

// Impostare il comando utilizzato per popolare l'elenco.
rowset.setCommand("SELECT * FROM MYLIB.DATABASE");

// Popolare il rowset.
rowset.execute();

// Aggiornare i bilanci del cliente.
while (rowset.next())
{
   double newBalance = rowset.getDouble("BALANCE") +
                       july_statements.getPurchases(rowset.getString("CUSTNUM"));
   rowset.updateDouble("BALANCE", newBalance);
   rowset.updateRow();
}
```

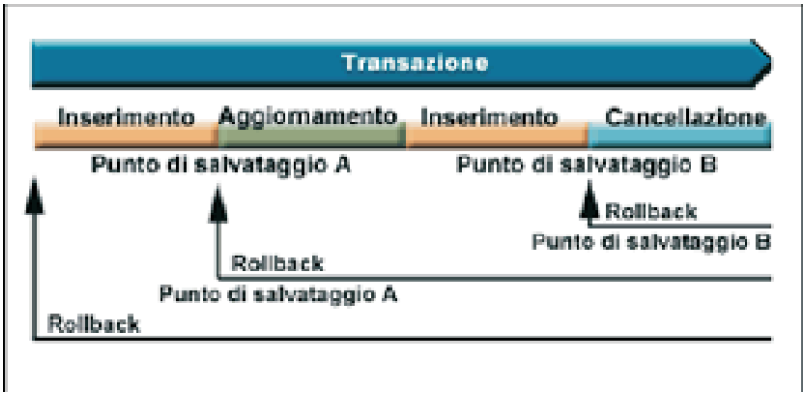
Esempio: creazione e popolamento di un oggetto AS400JDBCRowSet mentre si richiama il sorgente dati da JNDI

```
// Richiamare l'origine dati registrata in JNDI (presupporre che l'ambiente JNDI sia impostato).
Context context = new InitialContext();
AS400JDBCDataSource dataSource = (AS400JDBCDataSource) context.lookup("jdbc/customer");

AS400JDBCRowSet rowset = new AS400JDBCRowSet();
// Stabilire un collegamento impostando il nome dell'origine dati.
rowset.setDataSourceName("jdbc/customer");
rowset.setUsername("myuser");
rowset.setPassword("myPasswd");

// Impostare l'istruzione preparata e inizializzare i parametri.
rowset.setCommand("SELECT * FROM MYLIBRARY.MYTABLE WHERE STATE = ? AND BALANCE > ?");
rowset.setString(1, "MINNESOTA");
rowset.setDouble(2, MAXIMUM_LIMIT);

// Popolare il rowset.
rowset.execute();
```

Classe AS400JDBCSavepoint:

La classe AS400JDBCSavepoint rappresenta un punto di interruzione logico in una transazione. L'utilizzo dei punti di salvataggio fornisce un controllo più definito su quali modifiche sono interessate quando si effettua il roll back di una transazione.

Figura 1: utilizzo dei punti di salvataggio per controllare i rollback in una transazione



Ad esempio, Figura 1 mostra una transazione che include due punti di salvataggio, A e B. Effettuando il roll back della transazione in entrambi i punti di salvataggio si annullano (o invertono) solo quelle modifiche dal punto in cui un roll back viene richiamato in un punto di salvataggio. Ciò evita l'annullamento di tutte le modifiche nell'intera transazione. Tenere presente che una volta effettuato il rollback al punto di salvataggio A, non è possibile effettuare, successivamente, il rollback al punto di salvataggio B. Non è possibile accedere al punto di salvataggio B dopo aver effettuato il rollback del lavoro al suo interno.

Esempio: utilizzo dei punti di salvataggio

In questo scenario, si presume che l'applicazione aggiorni i record degli studenti. Al termine dell'aggiornamento di un determinato campo in ogni record studente, è possibile eseguire il commit. Il codice rileva un particolare errore associato all'aggiornamento di tale campo ed effettua il roll back del lavoro svolto quando si verifica questo errore. Questo particolare errore influisce solo sul lavoro svolto nel record corrente.

Quindi, si imposta un punto di salvataggio tra ogni aggiornamento dei record studente. A questo punto, quando si verifica tale errore, si effettua il rollback solo dell'ultimo aggiornamento nella tabella dello studente. Invece di effettuare il roll back di una grande quantità di lavoro, è possibile, ora, effettuare il roll back solo di una piccola parte di esso.

Il seguente codice di esempio serve ad illustrare come possono essere utilizzati i punti di salvataggio. L'esempio presuppone che l'ID studente di John sia 123456 e l'ID studente di Jane sia 987654.

```
// Richiamare un collegamento dall'unità di controllo
     Class.forName("com.ibm.as400.access.AS400JDBCDriver");

     // Richiamare un oggetto statement
         Statement statement = connection.createStatement();

     // Aggiornare il record di John con voto 'B' in ginnastica.
int rows = statement.executeUpdate(
  "UPDATE STUDENTTABLE SET GRADE_SECOND_PERIOD = 'B' WHERE STUDENT_ID= '123456'");

     // Impostare un punto di salvataggio contrassegnando un punto intermedio nella transazione
     Savepoint savepoint1 = connection.setSavepoint("SAVEPOINT_1");

     // Aggiornare il record di Jane con il voto 'C' in biochimica.
int rows = statement.executeUpdate(
   "UPDATE STUDENTTABLE SET GRADE_SECOND_PERIOD = 'C' WHERE STUDENT_ID= '987654'");

     // E' stato rilevato un errore, quindi è necessario eseguire il roll back del
     // record di Jane, ma non per quello di John.
     // Eseguire il rollback della transazione sul punto di salvataggio 1. La modifica al record di Jane
     // viene eliminata mentre la modifica al record di John rimane.
     connection.rollback(savepoint1);

     // Sincronizzare la transazione; solo il voto 'B' di John viene sincronizzato nel database.
     connection.commit();
```

Considerazioni e restrizioni

L'utilizzo dei punti di salvataggio richiede la conoscenza delle seguenti considerazioni e restrizioni:

Considerazioni

IBM Toolbox per Java segue le regole del database relative al modo in cui i rollback influiscono sui cursori e sui blocchi conservati. Ad esempio, quando si imposta l'opzione di collegamento per mantenere aperti i cursori dopo un rollback tradizionale, i cursori rimangono aperti anche dopo un rollback in un punto di salvataggio. In altre parole, quando una richiesta di rollback riguarda i punti di salvataggio, IBM Toolbox per Java non sposta o chiude il cursore quando il database sottostante non lo supporta.

L'utilizzo di un punto di salvataggio per effettuare il roll back di una transazione annulla solo le operazioni effettuate dal punto di partenza del roll back nel punto di salvataggio. Le operazioni eseguite prima del punto di salvataggio non subiscono cambiamenti. Come nell'esempio precedente, è necessario essere consapevoli che è possibile effettuare il commit di una transazione che includa il lavoro effettuato prima di un determinato punto di salvataggio ma non quello svolto dopo di esso.

Tutti i punti di salvataggio sono rilasciati e diventano non validi quando la transazione è sottoposta a commit o quando l'intera transazione è sottoposta a roll back. E' anche possibile rilasciare punti di salvataggio richiamando Connection.releaseSavepoint().

Restrizioni

Le seguenti restrizioni si applicano quando si utilizzano i punti di salvataggio:

- E' necessario che i punti di salvataggio abbiano nomi univoci.

- Non è possibile riutilizzare il nome di un punto di salvataggio fino a quando questo non viene rilasciato, sottoposto a commit o a roll back.
- E' necessario impostare l'auto-commit su 'OFF' per rendere validi i punti di salvataggio. E' possibile impostare l'auto-commit su 'OFF' utilizzando `Connection.setAutoCommit(false)`.L'attivazione dell'auto-commit durante l'utilizzo dei punti di salvataggio invia un'eccezione.
- I punti di salvataggio non sono validi attraverso i collegamenti XA. L'utilizzo di un collegamento XA con punti di salvataggio invia un'eccezione.
- E' necessario che il server esegua OS/400 Versione 5 Release 2 o versioni successive. L'utilizzo dei punti di salvataggio quando si effettua un collegamento (o si è già collegati) ad un server su cui è in esecuzione la V5R1 o versioni precedenti di OS/400 invia un'eccezione.

Descrizione lunga della Figura 1: utilizzo dei punti di salvataggio per controllare le operazioni di rollback in una transazione (rzahh586.gif):

In IBM Toolbox per Java: classe AS400JDBCSavepoint

Questa figura illustra come utilizzare punti di salvataggio per controllare le operazioni di rollback in una transazione.

Descrizione

La figura è composta da quanto segue:
- Una freccia orizzontale blu, che punta verso destra, etichettata 'Transazione'. La freccia Transazione rappresenta una transazione lineare che inizia sulla sinistra e termina sulla destra.
- Sotto la freccia Transazione vi è una barra multicolore di lunghezza pari alla freccia Transazione. La barra è divisa in quattro sezioni colorate che, da sinistra a destra, rappresenta azioni distinte che costituiscono la transazione. Sotto la barra vi sono due etichette che rappresentano i punti di salvataggio nella transazione.
- Sotto la barra multicolore vi sono tre frecce posizionate una sull'altra. Le frecce puntano verso sinistra ed ognuna rappresenta il rollback della transazione ad un punto differente.

La freccia Transazione rappresenta una transazione che inizia a sinistra e termina a destra. La transazione è composta da una serie di azioni distinte (le sezioni diverse della barra multicolore). Da sinistra a destra, le sezioni colorate rappresentano:
- La prima azione (una sezione rossiccia) etichettata 'Inserire'
- La seconda azione (una sezione verde) etichettata 'Aggiornare'
- La terza azione (un'altra sezione rossiccia) etichettata 'Inserire'
- la quarta e ultima azione (una sezione blu) etichettata 'Cancellare'

Le etichette al di sotto della barra multicolore rappresentano i punti di salvataggio. Il punto in cui termina la prima azione ed inizia la seconda è etichettato 'Punto di salvataggio A.' Il punto in cui termina la terza azione e inizia la quarta è etichettato 'Punto di salvataggio B.'

Le frecce al di sotto della barra multicolore puntano verso sinistra e indicano come i punti di salvataggio influenzino il rollback della transazione:
- La prima freccia punta su Punto di salvataggio B. Il rollback della transazione sul Punto di salvataggio B inverte solo l'azione finale (la sezione blu etichettata 'Cancellare')
- La seconda freccia punta su Punto di salvataggio A. Il rollback della transazione sul Punto di salvataggio A inverte le azioni dalla seconda alla quarta (la sezione verde etichettata 'Aggiornare', la seconda sezione rossiccia etichettata 'Inserire' e la sezione blu etichettata 'Cancellare')
- La freccia finale punta all'inizio della transazione. Il rollback completo della transazione inverte tutte le azioni distinte

Esecuzione di istruzioni SQL con oggetti Statement:

Utilizzare un oggetto Statement per eseguire un'istruzione SQL e facoltativamente ottenere il ResultSet da essa prodotto.

PreparedStatement eredita da Statement e CallableStatement eredita da PreparedStatement. Utilizzare i seguenti oggetti Statement per eseguire diverse istruzioni SQL:

- "Interfaccia Statement": esegue un'istruzione SQL semplice priva di parametri.
- "Interfaccia PreparedStatement" a pagina 74 - esegue un'istruzione SQL precompilata che potrebbe disporre o meno di parametri IN.
- "Interfaccia CallableStatement" a pagina 65 - esegue una chiamata a una procedura memorizzata nel database. Un CallableStatement può avere o meno parametri IN, OUT e INOUT.

L'oggetto Statement consente di inoltrare in un database più comandi SQL come un singolo gruppo tramite l'utilizzo del supporto batch. E' possibile ottenere prestazioni migliori utilizzando il supporto batch in quanto l'elaborazione di un gruppo di operazioni risulta solitamente più veloce rispetto all'elaborazione di un gruppo alla volta. Per ulteriori informazioni sull'utilizzo del supporto batch, consultare Potenziamenti del supporto JDBC.

Quando si utilizzano aggiornamenti batch, solitamente è necessario disattivare l'auto-commit. La disattivazione dell'auto-commit consente al programma di stabilire se sottoporre la transazione a commit se si è verificato un errore e non tutti i comandi sono stati eseguiti. In JDBC 2.0 e successive specificazioni di JDBC, un oggetto Statement può conservare la traccia di un elenco di comandi che possono essere inoltrati con esito positivo ed eseguiti insieme in un gruppo. Quando questo elenco di comandi batch viene eseguita dal metodo executeBatch(), i comandi vengono eseguiti nell'ordine nel quale essi sono stati aggiunti all'elenco.

AS400JDBCStatement fornisce i metodi che consentono di eseguire molte operazioni, incluse le seguenti:

- Eseguire diversi tipi di istruzioni
- Richiamare i valori di differenti parametri dell'oggetto Statement, inclusi:
  - Il collegamento
  - Ogni chiave auto-generata creata come risultato dell'esecuzione dell'oggetto Statement
  - La dimensione fetch e la direzione fetch
  - La massima dimensione del campo e il limite massimo della riga
  - La attuale impostazione risultati, la successiva impostazione risultati, il tipo di impostazione risultati, il risultato combinazione impostazione e il risultato supporto cursore impostazione
- Aggiungere una istruzione SQL al batch corrente
- eseguire il batch corrente delle istruzioni SQL

Interfaccia Statement

Utilizzare Connection.createStatement() per creare nuovi oggetti Statement.

Il seguente esempio mostra come utilizzare un oggetto Statement.

```
                              // Collegarsi al server.
    Connection c  = DriverManager.getConnection("jdbc:as400://mySystem");
                              // Creare un oggetto Statement.
         Statement s = c.createStatement();

                                          // Eseguire un'istruzione SQL che crea
                                          // una tabella nel database.
         s.executeUpdate("CREATE TABLE MYLIBRARY.MYTABLE (NAME VARCHAR(20), ID INTEGER)");

                                          // Eseguire un'istruzione SQL che inserisca
```

```
                                        // un record nella tabella.
    s.executeUpdate("INSERT INTO MYLIBRARY.MYTABLE (NAME, ID) VALUES ('DAVE', 123)");

                                        // Eseguire un'istruzione SQL che inserisca
                                        // un record nella tabella.
    s.executeUpdate("INSERT INTO MYLIBRARY.MYTABLE (NAME, ID) VALUES ('CINDY', 456)");

                              // Eseguire un'interrogazione SQL sulla tabella.
ResultSet rs = s.executeQuery("SELECT * FROM MYLIBRARY.MYTABLE");

                                        // Chiudere Statement e
                                        // Connection.
    s.close();
c.close();
```

JDBC XA Distributed Transaction Management:

Le classi JDBC XA distributed transaction management consentono di utilizzare l'unità di controllo JDBC di IBM Toolbox per Java in una transazione distribuita. L'utilizzo delle classi XA per abilitare l'unità di controllo JDBC di IBM Toolbox per Java consente di partecipare a transazioni che comprendono più sorgenti di dati.

Generalmente, le classi XA distributed transaction management vengono utilizzate e controllate direttamente dal gestore transazioni, il quale è separato dall'unità di controllo JDBC. Le interfacce di gestione delle transazioni distribuite vengono definite come parte del JDBC 2.0 Optional Package e della JTA (Java Transaction API). Entrambi sono disponibili tramite Sun come file jar. Le interfacce di gestione delle transazioni distribuite sono anche supportate nell'API JDBC 3.0, collegata a Java 2 Platform, Standard Edition, versione 1.4.

Per ulteriori informazioni, consultare i siti web della Sun per JDBC e JTA.

Utilizzare i seguenti oggetti per consentire all'unità di controllo JDBC di IBM Toolbox per Java di partecipare alle transazioni distribuite XA:

- AS400JDBCXADataSource - Un sistema di produzione per oggetti AS400JDBCXAConnection.Questa è una sottoclasse di AS400JDBCDataSource.
- AS400JDBCXACConnection - Un oggetto del lotto di collegamenti che fornisce hook per le gestione del lotto di collegamenti e la gestione delle risorse di XA.
- AS400JDBCXAResource - Un gestore risorse da utilizzare nella gestione delle transazioni XA.

Esempio: utilizzo delle classi XA

Il seguente esempio illustra un utilizzo di tipo semplice delle classi XA. E' necessario ricordare che i dettagli dovrebbero essere completati tramite il lavoro che utilizza altre sorgenti dati. Questo tipo di codice viene visualizzato, generalmente, in un gestore di transazioni.

```
  // Creare un'origine dati XA per stabilire un collegamento XA.
AS400JDBCXADataSource xaDataSource = new AS400JDBCXADataSource("myAS400");
xaDataSource.setUser("myUser");
xaDataSource.setPassword("myPasswd");

// Richiamare un XAConnection e l'XAResource associato.
// Ciò fornisce l'accesso al gestore risorse.
XAConnection xaConnection = xaDataSource.getXAConnection();
XAResource xaResource = xaConnection.getXAResource();

// Generare un nuovo Xid (scelto dal gestore transazioni).
Xid xid = ...;

// Avviare la transazione.
xaResource.start(xid, XAResource.TMNOFLAGS);
```

```
        // ...Gestire il database...

        // Fine della transazione.
        xaResource.end(xid, XAResource.TMSUCCESS);

        // Preparare per la sincronizzazione.
        xaResource.prepare(xid);

        // Sincronizzare la transazione.
        xaResource.commit(xid, false);

    // Chiudere il collegamento XA quando terminato. Questa operazione chiude
        // esplicitamente la risorsa XA.
        xaConnection.close();
```

Classi Jobs

Le classi Jobs di IBM Toolbox per Java (nel pacchetto Access) consentono al programma Java di richiamare e modificare le informazioni sul lavoro.

Nota: IBM Toolbox per Java fornisce inoltre le classi resource che forniscono una framework generica e un'interfaccia di programmazione coerente per la gestione di vari oggetti ed elenchi iSeries. Dopo aver letto informazioni relative alle classi nel pacchetto access e nel pacchetto resource, è possibile scegliere l'oggetto più adatto all'applicazione. Le classi resource per gestire i lavori sono RJob, RJobList e RJobLog.

Utilizzare le classi Jobs per gestire i seguenti tipi di informazioni sul lavoro:
- Informazioni data/ora
- Coda lavori
- Identificativi lingua
- Registrazione messaggi
- Coda emissioni
- Informazioni stampante

Le classi Jobs nel pacchetto Access sono le seguenti:
- Job - richiama e modifica le informazioni sul lavoro iSeries
- JobList - richiama un elenco di lavori iSeries
- JobLog - rappresenta la registrazione lavori di un iSeries

Esempi

Il seguente esempio mostra vari metodi di utilizzo delle classi Job, JobList e JobLog. Il primo esempio mostra un metodo di utilizzo di una memoria cache con la classe Job. I collegamenti ad altri esempi si trovano subito dopo il codice di esempio.

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per informazioni legali importanti.

Esempio: utilizzo di una memoria cache durante l'impostazione di un valore e il richiamo di un valore

```
        try {

          // Crea un oggetto AS400.
          AS400 as400 = new AS400("systemName");

          // Crea un oggetto Job
          Job job = new Job(as400,"QDEV002");

          // Richiama le informazioni lavoro
          System.out.println("User of this job :" + job.getUser());
```

```
            System.out.println("CPU used :" + job.getCPUUsed();
            System.out.println("Job enter system date : " + job.getJobEnterSystemDate());

            // Imposta la modalità cache
            job.setCacheChanges(true);

            // Le modifiche verranno memorizzate nella cache.
            job.setRunPriority(66);
            job.setDateFormat("*YMD");

     // Sincronizzare le modifiche. Questa operazione modificherà il valore su iSeries.
            job.commitChanges();

            // Impostare le informazioni sul lavoro direttamente nel sistema(senza cache).
            job.setCacheChanges(false);
            job.setRunPriority(60);
   } catch (Exception e)
{
            System.out.println(quot;error :" + e)
}
```

I seguenti esempi mostrano come elencare i lavori appartenenti a un utente specifico, come elencare i lavori con le informazioni sullo stato del lavoro e visualizzare i messaggi in una registrazione lavori:

"Esempio: utilizzo di JobList per elencare le informazioni di identificazione del lavoro" a pagina 508

"Esempio: utilizzo di JobList per richiamare un elenco di lavori" a pagina 510

"Esempio: utilizzo di JobLog per visualizzare messaggi nella registrazione lavori" a pagina 513

Classe Job:

La classe Job (nel pacchetto di accesso) consente a un programma Java di richiamare e modificare le informazioni relative al lavoro server.

Nota: IBM Toolbox per Java inoltre fornisce le classi resource che forniscono una framework generica un'interfaccia di programmazione coerente per la gestione di vari oggetti ed elenchi sul server iSeries. Dopo aver letto informazioni relative alle classi nel pacchetto access e nel pacchetto resource, è possibile scegliere l'oggetto più adatto all'applicazione. Le classi resource per gestire i lavori sono RJob, RJobList e RJobLog.

E' possibile richiamare e modificare il seguente tipo di informazioni sul lavoro tramite la classe Job:

- Code lavori
- Code di emissioni
- Registrazione messaggi
- Unità di stampa
- Identificativo Paese o regione
- Formato data

La classe Job consente anche di modificare un singolo valore per volta o memorizzare nella cache numerose modifiche utilizzando il metodosetCacheChanges(true) ed eseguire il commit delle modifiche utilizzando il metodocommitChanges().Se la memorizzazione in cache non è attiva, non è necessario eseguire il commit.

Esempio

Per un esempio di codice, consultare la documentazione di riferimento Javadoc per la classe Job. Questo esempio mostra come impostare e richiamare i valori per e dalla memoria cache in modo da impostare la priorità di esecuzione con il metodo setRunPriority() ed impostare il formato data con il metodo setDateFormat():

Lavoro

Classe JobList:

E' possibile utilizzare la classe JobList (nel pacchetto Access) per elencare i lavori iSeries.

Nota: IBM Toolbox per Java fornisce inoltre le classi resource che forniscono una framework generica e un'interfaccia di programmazione coerente per la gestione di vari oggetti ed elenchi iSeries. Dopo aver letto informazioni relative alle classi nel pacchetto access e nel pacchetto resource, è possibile scegliere l'oggetto più adatto all'applicazione. Le classi resource per gestire i lavori sono RJob, RJobList e RJobLog.

Con la classe JobList, è possibile richiamare quanto segue:
- Tutti i lavori
- Lavori per nome, numero lavoro o utente

Utilizzare il metodo getJobs() per visualizzare un elenco di lavori iSeries o il metodo getLength() per visualizzare il numero di lavori richiamati con l'ultimo metodo getJobs().

Esempio: utilizzo di JobList

Il seguente esempio elenca tutti i lavori attivi nel sistema:

```
                    // Creare un oggetto AS400.
Elencare i
                                          // lavori su questo iSeries.
     AS400 sys = new AS400("mySystem.myCompany.com");

                                          // Creare l'oggetto elenco lavori.
          JobList jobList = new JobList(sys);

                                          // Richiamare l'elenco di lavori attivi.
              Enumeration list = jobList.getJobs();

                                          // Per ogni lavoro attivo sul questo sistema
                                          // vengono stampate le informazioni sul lavoro.
              while (list.hasMoreElements())
     {
                    Job j = (Job) list.nextElement();

         System.out.println(j.getName() + "." +
                           j.getUser() + "." +
                                                          j.getNumber());
     }
```

Classe JobLog:

La classe JobLog (nel pacchetto Access) richiama i messaggi nella registrazione lavori di un lavoro server tramite la chiamata a getMessages().

Nota: IBM Toolbox per Java fornisce inoltre le classi resource che forniscono una framework generica e un'interfaccia di programmazione coerente per la gestione di vari oggetti ed elenchi iSeries. Dopo aver letto informazioni relative alle classi nel pacchetto access e nel pacchetto resource, è possibile scegliere l'oggetto più adatto all'applicazione. Le classi resource per gestire i lavori sono RJob, RJobList e RJobLog.

Esempio: utilizzo di JobLog

Nel seguente esempio vengono stampati tutti i messaggi nella registrazione lavori per l'utente specificato:

```
                                        // ... Impostare il lavoro per creare un oggetto AS400
                                        // mentre un oggetto jobList è già stato
                                        // impostato

                                        // Richiamare l'elenco di lavori attivi
                                        // su iSeries
          Enumeration list = jobList.getJobs();

                                        // Consultare l'elenco per trovare un
                                        // lavoro per l'utente specificato.
          while (list.hasMoreElements())
{
               Job j = (Job) list.nextElement();

            if (j.getUser().trim().equalsIgnoreCase(userID))
   {
                                        // E' stato rilevato un lavoro che corrisponde
                 // all'utente corrente. Creare un oggetto
                                        // registrazione lavori per questo lavoro.
      JobLog jlog = new JobLog(system, j.getName(), j.getUser(), j.getNumber());

                                        // Numerare i messaggi nella registrazione
                                        // lavori quindi stamparli.
                Enumeration messageList = jlog.getMessages();

                while (messageList.hasMoreElements())
      {
                        AS400Message message = (AS400Message) messageList.nextElement();
             System.out.println(message.getText());
      }

   }
}
```

Classi Message AS400Message

L'oggetto AS400Message consente al programma Java di richiamare un messaggio OS/400 creato da una precedente operazione (ad esempio, da una chiamata al comando). Da un oggetto message, il programma Java può richiamare quanto segue:

- La libreria iSeries javadoc/com/ibm/as400/access/AS400Message.html#GETLIBRARYNAME() e il file messaggi che contengono il messaggio
- L' ID del messaggio
- Il tipo di messaggio
- La severità del messaggio
- Il testo del messaggio
- Il testo di aiuto del messaggio

Il seguente esempio mostra come utilizzare l'oggetto AS400Message:

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per informazioni legali importanti.

```
                  // Creare un oggetto chiamata al comando.
CommandCall cmd = new CommandCall(sys, "myCommand");

                  // Eseguire il comando
cmd.run();

                  // Richiamare l'elenco dei messaggi che costituiscono
```

```
                 // il risultato del comando che è stato
                 // appena eseguito
AS400Message[] messageList = cmd.getMessageList();

                // Iterare l'elenco
                // visualizzando i messaggi
for (int i = 0; i < messageList.length; i++)
{
   System.out.println(messageList[i].getText());
}
```

Esempi: utilizzo degli elenchi di messaggi

I seguenti esempi mostrano come è possibile utilizzare elenchi di messaggi con CommandCall e ProgramCall.

- "Esempio: utilizzo di CommandCall" a pagina 473
- "Esempio: utilizzo di ProgramCall" a pagina 525

QueuedMessage

La classe QueuedMessage estende la classe AS400Message.

Nota: Toolbox per Java fornisce inoltre le classi resource che forniscono una framework generica e un'interfaccia di programmazione coerente per la gestione di vari oggetti ed elenchi iSeries. Dopo aver letto informazioni relative alle classi nel pacchetto access e nel pacchetto resource, è possibile scegliere l'oggetto più adatto all'applicazione. La classe resourse per gestire i messaggi in coda è RQueuedMessage.

La classe QueuedMessage accede alle informazioni relative a un messaggio su una coda messaggi iSeries. Con questa classe, un programma Java può richiamare:

- Informazioni sull'origine del messaggio, ad esempio programma, nome lavoro, numero lavoro e utente
- La coda messaggi
- La chiave del messaggio
- Lo stato di risposta del messaggio

Il seguente esempio stampa tutti i messaggi nella coda messaggi dell'utente corrente (collegato):

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per informazioni legali importanti.

```
                // La coda messaggi si trova su questo iSeries.
AS400 sys = new AS400(mySystem.myCompany.com);

                // Creare l'oggetto coda messaggi.
                // Questo oggetto rappresenterà la
                // coda per l'utente corrente.
MessageQueue queue = new MessageQueue(sys, MessageQueue.CURRENT);

                // Richiamare l'elenco di messaggi attualmente
                // presenti in questa coda dell'utente.
Enumeration e = queue.getMessages();

                // Stampare ogni messaggio nella coda.
while (e.hasMoreElements())
{
   QueuedMessage msg = e.getNextElement();
   System.out.println(msg.getText());
}
```

MessageFile

La classe MessageFile consente di ricevere un messaggio da un file di messaggi iSeries. La classe MessageFile restituisce un oggetto AS400Message che contiene il messaggio. Utilizzando la classe MessageFile, è possibile effettuare le seguenti operazioni:

- Restituire un oggetto message che contiene il messaggio
- Restituire un oggetto message che contiene iltesto di sostituzione nel messaggio

Il seguente esempio mostra come richiamare e stampare un messaggio:

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per informazioni legali importanti.

```
AS400 system = new AS400("mysystem.mycompany.com");
MessageFile messageFile = new MessageFile(system);
messageFile.setPath("/QSYS.LIB/QCPFMSG.MSGF");
AS400Message message = messageFile.getMessage("CPD0170");
System.out.println(message.getText());
```

MessageQueue

La classe MessageQueue consente a un programma Java di interagire con una coda messaggi iSeries.

Nota: Toolbox per Java fornisce inoltre le classi resource che forniscono una framework generica e un'interfaccia di programmazione coerente per la gestione di vari oggetti ed elenchi iSeries. Dopo aver letto informazioni relative alle classi nel pacchetto access e nel pacchetto resource, è possibile scegliere l'oggetto più adatto all'applicazione.La classe resource per gestire le code messaggi è RMessageQueue.

La classe MessageQueue funge da contenitore per la classe QueuedMessage. Il metodo getMessages(), in particolare, restituisce un elenco di oggetti QueuedMessage.La classe MessageQueue può eseguire queste operazioni:

- Impostare gli attributi della coda messaggi
- Richiamare informazioni su una coda messaggi
- Ricevere messaggi da una coda messaggi
- Inviare messaggi ad una coda messaggi
- Rispondere ai messaggi

Il seguente esempio elenca i messaggi nella coda messaggi per l'utente corrente:

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per informazioni legali importanti.

```
                // La coda messaggi si trova su questo iSeries.
AS400 sys = new AS400(mySystem.myCompany.com);

                // Creare l'oggetto coda messaggi.
                // Questo oggetto rappresenterà la
                // coda per l'utente corrente.
MessageQueue queue = new MessageQueue(sys, MessageQueue.CURRENT);

                // Richiamare l'elenco di messaggi attualmente
                // presenti in questa coda dell'utente.
Enumeration e = queue.getMessages();

                // Stampare ogni messaggio nella coda.
while (e.hasMoreElements())
{
   QueuedMessage msg = e.getNextElement();
   System.out.println(msg.getText());
}
```

NetServer

La classe NetServer rappresenta il servizio NetServer su un server iSeries. Gli oggetti NetServer consentono di interrogare e modificare lo stato e la configurazione del servizio NetServer.

Ad esempio, è possibile utilizzare la classe NetServer per:
- Avviare o arrestare il NetServer
- Richiamare un elenco di tutte le condivisioni di file e condivisioni di stampa
- Richiamare un elenco di tutte le sessioni correnti
- Interrogare e modificare i valori di attributo (utilizzando i metodi ereditati da ChangeableResource)

Nota: per utilizzare la classe NetServer, è necessario disporre di un profilo utente del server con autorizzazione *IOSYSCFG.

La classe NetServer è un'estensione di ChangeableResource e Resource, in modo da fornire una raccolta di ″attributi″ per rappresentare i vari valori ed impostazioni di NetServer.Interrogare o modificare gli attributi per accedere a o modificare la configurazione di NetServer.Alcuni degli attributi di NetServer sono:
- NAME
- NAME_PENDING
- DOMAIN
- ALLOW_SYSTEM_NAME
- AUTOSTART
- CCSID
- WINS_PRIMARY_ADDRESS

Attributi in sospeso

Molti degli attributi di NetServer sono in sospeso (ad esempio, NAME_PENDING).Gli attributi in sospeso rappresentano i valori di NetServer che si attivano al successivo avvio (o riavvio) di NetServer sul server.

Quando è disponibile una coppia di attributi correlati e un attributo è in sospeso mentre l'altro non lo è:
- L'attributo in sospeso è lettura/scrittura, quindi è possibile modificarlo
- L'attributo non in sospeso è di sola lettura, quindi è possibile interrogarlo ma non modificarlo

Altre classi NetServer

Le classi NetServer correlate consentono di richiamare ed impostare informazioni dettagliate su collegamenti specifici, sessioni, condivisioni di file e condivisioni di stampa:
- NetServerConnection: rappresenta un collegamento NetServer
- NetServerFileShare: rappresenta una condivisione del server file NetServer
- NetServerPrintShare: rappresenta una condivisione del server di stampa NetServer
- NetServerSession: rappresenta una sessione NetServer
- NetServerShare: rappresenta una condivisione NetServer

Esempio: utilizzo di un oggetto NetServer per modificare il nome del NetServer

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per informazioni legali importanti.

```
// Creare un oggetto di sistema che rappresenti il server iSeries.
AS400 system = new AS400("MYSYSTEM", "MYUSERID", "MYPASSWD");
```

```
// Creare un oggetto con cui interrogare e modificare NetServer.
NetServer nServer = new NetServer(system);

// Impostare "pending name" su NEWNAME.
nServer.setAttributeValue(NetServer.NAME_PENDING, "NEWNAME");

// Sincronizzare le modifiche. Questa operazione invia le modifiche al server.
nServer.commitAttributeChanges();

// Il nome NetServer rimarrà impostato su NEWNAME alla successiva
// chiusura e riavvio di NetServer.
```

Classi Permission

Le classi Permission consentono di richiamare e impostare le informazioni sull'autorizzazione all'oggetto. Le informazioni sull'autorizzazione dell'oggetto sono anche note come permessi. La classe Permission rappresenta una raccolta di autorizzazioni di molti utenti ad un oggetto specifico. La classe UserPermission rappresenta l'autorizzazione di un singolo utente ad uno specifico oggetto.

Classe Permission

La classe Permission consente di richiamare e modificare le informazioni sull'autorizzazione all'oggetto.Comprende una raccolta di numerosi utenti autorizzati all'oggetto. L'oggetto Permission consente al programma Java di memorizzare nella cache le modifiche all'autorizzazione fino a quando non viene chiamato il metodo commit(). Una volta chiamato il metodo commit(), tutte le modifiche effettuate fino a quel momento vengono inviate al server. Alcune delle funzioni fornite dalla classe Permission includono:

- addAuthorizedUser(): aggiunge un utente autorizzato.
- commit(): convalida le modifiche ai permessi nel server.
- getAuthorizationList(): restituisce l'elenco delle autorizzazioni dell'oggetto.
- getAuthorizedUsers(): restituisce una enumerazione degli utenti autorizzati.
- getOwner(): restituisce il nome del proprietario dell'oggetto.
- getSensitivityLevel(): restituisce il livello di sensibilità dell'oggetto.
- getType(): restituisce il tipo di autorizzazione all'oggetto (QDLO, QSYS o Root).
- getUserPermission(): restituisce il permesso di un utente specifico all'oggetto.
- getUserPermissions(): restituisce una enumerazione di permessi degli utenti all'oggetto.
- setAuthorizationList(): imposta l'elenco delle autorizzazioni relative all'oggetto.
- setSensitivityLevel(): imposta il livello di sensibilità dell'oggetto.

Esempio: utilizzo di Permission

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

Il seguente esempio mostra come creare un permesso ed aggiungere un utente autorizzato ad un oggetto.

```
// Creare l'oggetto AS400
AS400 as400 = new AS400();

// Creare l'autorizzazione inoltrando l'AS400 e l'oggetto
Permission myPermission = new Permission(as400, "QSYS.LIB/myLib.LIB");

// Aggiungere un utente da autorizzare all'oggetto
myPermission.addAuthorizedUser("User1");
```

Classe UserPermission

La classe UserPermission rappresenta l'autorizzazione di un singolo, specifico utente.UserPermission comprende tre sottoclassi che gestiscono l'autorizzazione in base al tipo di oggetto:

Classe UserPermission	Descrizione
DLOPermission	Rappresenta l'autorizzazione di un utente ai DLO (Document Library Objects), memorizzati in QDLS.
QSYSPermission	Rappresenta l'autorizzazione di un utente agli oggetti memorizzati in QSYS.LIB e contenuti nel server.
RootPermission	Rappresenta l'autorizzazione di un utente agli oggetti contenuti nella struttura dell'indirizzario principale. Gli oggetti RootPermissions sono quegli oggetti non contenuti in QSYS.LIB o QDLS.

La classe UserPermission consente di svolgere le seguenti operazioni:

- Determinare se il profilo utente è un profilo di gruppo
- Restituire il nome del profilo utente
- Indicare se l'utente dispone dell'autorizzazione
- Impostare l'autorizzazione della gestione dell'elenco di autorizzazioni

Esempio: utilizzo di UserPermission

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

Il seguente esempio mostra come richiamare utenti e gruppi che dispongono del permesso relativo ad un oggetto e stamparli uno alla volta.

```
            // Creare un oggetto di sistema.
            AS400 sys = new AS400("MYAS400", "USERID", "PASSWORD");

            // Rappresentare le autorizzazioni ad un oggetto sul sistema, come ad esempio una libreria.
            Permission objectInQSYS = new Permission(sys, "/QSYS.LIB/FRED.LIB");

            // Reperire diversi utenti/gruppi che hanno autorizzazioni impostate su tale oggetto.
            Enumeration enum = objectInQSYS.getUserPermissions();
                  while (enum.hasMoreElements ())
      {
                // Stampare i nomi profilo utente/di gruppo uno alla volta.
                UserPermission userPerm = (UserPermission)enum.nextElement();
                System.out.println(userPerm.getUserID());
      }
```

Classe DLOPermission:

DLOPermission è una sottoclasse di UserPermission. DLOPermission consente di visualizzare ed impostare le autorizzazioni di cui un utente dispone per un DLO (Document Library Object).

Uno dei seguenti valori di autorizzazione viene assegnato ad ogni utente.

Valore autorizzazione	Descrizione
*ALL	L'utente può eseguire tutte le operazioni tranne quelle controllate dalla gestione elenchi di autorizzazioni.
*AUTL	L'elenco di autorizzazioni viene utilizzato per determinare l'autorizzazione al documento.

Valore autorizzazione	Descrizione
*CHANGE	L'utente può modificare ed eseguire funzioni di base sull'oggetto.
*EXCLUDE	L'utente non può accedere all'oggetto.
*USE	L'utente dispone dell'autorizzazione operativa all'oggetto, dell'autorizzazione alla lettura e dell'autorizzazione all'esecuzione.

E' necessario che l'utente utilizzi uno dei seguenti metodi per modificare o determinare l'autorizzazione dell'utente:

- Utilizzare getDataAuthority() per visualizzare il valore dell'autorizzazione dell'utente
- Utilizzare setDataAuthority() per impostare il valore dell'autorizzazione dell'utente

Dopo aver impostato le autorizzazioni, è importante utilizzare il metodo commit() dalla classe Autorizzazioni per inviare le modifiche al server.

Per ulteriori informazioni su autorizzazioni e permessi, consultare il capitolo 5: Sicurezza delle risorse nel manuale **Riferimenti alla sicurezza iSeries**  .

Esempio: utilizzo di DLOPermission

Il seguente esempio mostra come richiamare e stampare le autorizzazioni DLO, compresi i profili utente per ciascuna autorizzazione.

```
// Creare un oggetto di sistema.

AS400 sys = new AS400("MYAS400", "USERID", "PASSWORD");
// Ripresentare i permessi ad un oggetto DLO.
Permission objectInQDLS = new Permission(sys, "/QDLS/MyFolder");

// Stampare il pathname oggetto e richiamarne i permessi.
System.out.println("Permissions on " + objectInQDLS.getObjectPath() + " are as follows:");
Enumeration enum = objectInQDLS.getUserPermissions();
while (enum.hasMoreElements ())
{
  // Per ciascuno dei permessi, stampare il nome profilo utente
  // e i relativi permessi utente all'oggetto.
  DLOPermission dloPerm = (DLOPermission)enum.nextElement();
  System.out.println(dloPerm.getUserID() + ": " + dloPerm.getDataAuthority());
}
```

QSYSPermission:

QSYSPermission è una sottoclasse della classe UserPermission.QSYSPermission consente di visualizzare ed impostare l'autorizzazione di cui un utente dispone per un oggetto nella struttura di libreria tradizionale di iSeries memorizzata in QSYS.LIB. E' possibile impostare l'autorizzazione per un oggetto memorizzato in QSYS.LIB impostando un valore di autorizzazione definito dal sistema o impostando le singole autorizzazioni per gli oggetti e per i dati.

La seguente tabella elenca e descrive i valori di autorizzazioni definiti dal sistema validi:

Valore di autorizzazione definiti dal sistema	Descrizione
*ALL	L'utente può eseguire tutte le operazioni tranne quelle controllate dalla gestione elenchi di autorizzazioni.
*AUTL	L'elenco di autorizzazioni è utilizzato per determinare l'autorizzazione al documento.

Valore di autorizzazione definiti dal sistema	Descrizione
*CHANGE	L'utente può modificare ed eseguire funzioni di base sull'oggetto.
*EXCLUDE	L'utente non può accedere all'oggetto.
*USE	L'utente dispone dell'autorizzazione operativa all'oggetto, dell'autorizzazione alla lettura e dell'autorizzazione all'esecuzione.

Ogni valore di autorizzazione definito dal sistema rappresenta in realtà una combinazione delle singole autorizzazioni per gli oggetti e per i dati. La seguente tabella illustra le relazioni delle autorizzazioni definite dal sistema con le singole autorizzazioni per gli oggetti e per i dati:

Tabella 1. **Y** *fa riferimento alle autorizzazioni che è possibile assegnare. n fa riferimento alle autorizzazioni che non è possibile assegnare.*

Autorizzaz. definita dal sistema	Autorizzazione per gli oggetti					Autorizzazione per i dati				
	Opr	Mgt	Exist	Alter	Ref	Read	Add	Upd	Dlt	Exe
All	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**
Change	**Y**	n	n	n	n	**Y**	**Y**	**Y**	**Y**	**Y**
Exclude	n	n	n	n	n	n	n	n	n	n
Use	**Y**	n	n	n	n	**Y**	n	n	n	**Y**
Autl	Valido solo per utente (*PUBLIC) e per un elenco di autorizzazioni specifico che determina le singole autorizzazioni per gli oggetti e per i dati.									

Specificando un'autorizzazione definita dal sistema si assegnano automaticamente le singole autorizzazioni appropriate. Allo stesso modo, specificando le diverse singole autorizzazioni si modificano i valori corretti delle singole autorizzazioni. Quando una combinazione di singole autorizzazioni per gli oggetti e per i dati non corrisponde a un singolo valore di autorizzazione definito dal sistema, il singolo valore diventa ″Definito dall'utente.″

Utilizzare il metodo getObjectAuthority() per visualizzare l'autorizzazione corrente definita dal sistema.Utilizzare il metodo setObjectAuthority() per impostare l'autorizzazione corrente definita dal sistema utilizzando un singolo valore.

Utilizzare il metodo Set appropriato per impostare su on o su off i valori delle singole autorizzazioni agli oggetti:

- setAlter()
- setExistence()
- setManagement()
- setOperational()
- setReference()

Utilizzare il metodo Set appropriato per impostare su on o su off i valori delle singole autorizzazioni ai dati:

- setAdd()
- setDelete()
- setExecute()
- setRead()
- setUpdate()

Per ulteriori informazioni relative alle differenti autorizzazioni, consultare il capitolo 5: Sicurezza delle risorse nel manuale **Riferimenti alla sicurezza iSeries**  .Per informazioni sull'utilizzo dei comandi CL di iSeries per l'assegnazione e la modifica delle autorizzazioni agli oggetti, consultare i comandi CL di iSeries GRTOBJAUT (Assegnazione autorizzazione oggetto) e EDTOBJAUT (Modifica autorizzazione oggetto).

Esempio

Questo esempio mostra in che modo richiamare e stampare le autorizzazioni per un oggetto QSYS.

```
        // Creare un oggetto di sistema.
        AS400 sys = new AS400("MYAS400", "USERID", "PASSWORD");

        // Rappresentare le autorizzazioni ad un oggetto QSYS.
        Permission objectInQSYS = new Permission(sys, "/QSYS.LIB/FRED.LIB");

        // Stampare il pathname oggetto e richiamarne i permessi.
        System.out.println("Permissions on "+objectInQSYS.getObjectPath()+" are as follows:");
        Enumeration enum = objectInQSYS.getUserPermissions();
              while (enum.hasMoreElements ())
{
            // Per ciascuno dei permessi, stampare il nome profilo utente
            // e i relativi permessi utente all'oggetto.
            QSYSPermission qsysPerm = (QSYSPermission)enum.nextElement();
            System.out.println(qsysPerm.getUserID()+": "+qsysPerm.getObjectAuthority());
}
```

RootPermission:

RootPermission è una sottoclasse della classe UserPermission.La classe RootPermission consente di visualizzare ed impostare le autorizzazioni per l'utente di un oggetto contenuto in una struttura di indirizzario principale.

Un oggetto nella struttura di indirizzario principale può impostare l'autorizzazione ai dati o l'autorizzazione all'oggetto. E' possibile impostare l'autorizzazione ai dati sui valori elencati nella tabella che segue. Utilizzare il metodo getDataAuthority() per visualizzare i valori correnti e il metodo setDataAuthority() per impostare l'autorizzazione ai dati.

La tabella che segue elenca e descrive i valori validi di autorizzazione ai dati:

Valore di autorizzazione ai dati	**Descrizione**
*none	L'utente non ha alcuna autorizzazione all'oggetto
*RWX	L'utente ha l'autorizzazione a leggere, aggiungere, aggiornare, cancellare ed eseguire.
*RW	L'utente ha l'autorizzazione a leggere, aggiungere e cancellare.
*RX	L'utente ha l'autorizzazione a leggere ed eseguire.
*WX	L'utente ha l'autorizzazione ad aggiungere, aggiornare, cancellare ed eseguire.
*R	L'utente ha l'autorizzazione a leggere.
*W	L'utente ha l'autorizzazione ad aggiungere, aggiornare e cancellare.
*X	L'utente ha l'autorizzazione ad eseguire.
*EXCLUDE	L'utente non può accedere all'oggetto.
*AUTL	Le autorizzazioni pubbliche per questo oggetto provengono dall'elenco di autorizzazioni.

L'autorizzazione all'oggetto può essere impostata su uno o più dei valori che seguono: alter, existence, management o reference. E' possibile utilizzare i metodi setAlter(), setExistence(), setManagement() o setReference() per impostare i valori su on oppure su off.

Dopo aver impostato l'autorizzazione ai dati o l'autorizzazione all'oggetto, è importante utilizzare il metodo commit() dalla classe Permissions per inviare le modifiche al server.

Per ulteriori informazioni relative alle differenti autorizzazioni, consultare il capitolo 5: Sicurezza delle risorse nel manuale **Riferimenti alla sicurezza iSeries**  .

Esempio

L'esempio mostra come richiamare e stampare le autorizzazioni per un oggetto principale.

```
      // Creare un oggetto di sistema.
      AS400 sys = new AS400("MYAS400", "USERID", "PASSWORD");

      // Rappresentare i permessi ad un oggetto nel file system principale.
Permission objectInRoot = new Permission(sys, "/fred");

      // Stampare il pathname oggetto e richiamarne i permessi.
System.out.println("Permissions on "+objectInRoot.getObjectPath()+" are as follows:");
      Enumeration enum = objectInRoot.getUserPermissions();
            while (enum.hasMoreElements ())
{
          // Per ciascuno dei permessi, stampare il nome profilo utente
          // e i relativi permessi utente all'oggetto.
          RootPermission rootPerm = (RootPermission)enum.nextElement();
  System.out.println(rootPerm.getUserID()+": "+rootPerm.getDataAuthority());
}
```

Classi Print

Gli oggetti di stampa includono i file di spool, le code di emissione, le stampanti, i file di stampa, i lavori di scrittura e le risorse AFP (Advanced Function Printing), che includono i font, le definizioni di formato, sostituzioni, definizioni di pagina e segmenti di pagina. Le risorse AFP sono accessibili solo sulla Versione 3 Rilascio 7 (V3R7) e versioni successive di OS/400. (Il tentativo di aprire un AFPResourceList su un sistema che esegue una versione precedente di V3R7 crea una eccezione RequestNotSupportedException.)

Le classi IBM Toolbox per Java per gli oggetti di stampa sono organizzati su una classe di base, PrintObject e su una sottoclasse per ognuno dei sei tipi di oggetti di stampa.La classe di base contiene i metodi e gli attributi comuni a tutti gli oggetti di stampa del server. Le sottoclassi contengono i metodi e gli attributi specifici ad ogni tipo secondario.

Utilizzare le classi Print per svolgere le seguenti operazioni:

- Gestire gli oggetti di stampa del server:
  - Classe PrintObjectList - utilizzare per elencare e gestire gli oggetti di stampa del server.(Gli oggetti di stampa includono i file di spool, le code di emissione, le stampanti, le risorse AFP (Advanced Function Printing), i file di stampa ed i lavori del programma di scrittura)
  - Classe di base PrintObject - utilizzare per gestire gli oggetti di stampa
- Richiamo attributi PrintObject
- Creazione di nuovi file di spool del server utilizzando la classe SpooledFileOutputStream (utilizzare per i dati di stampa basati su EBCDIC)
- Creazione di flussi dati di stampa SCS (SNA Character Stream)
- Lettura dei file di spool e delle risorse AFP utilizzando PrintObjectInputStream
- Lettura dei file di spool utilizzando PrintObjectPageInputStream e PrintObjectTransformedInputStream
- Copiare i file di spool

- Visualizzare i file di spool AFP e SCS

Esempi

- Esempio: creazione di file di spool mostra come creare un file di spool su un server da un flusso di immissione
- Esempio: creazione di file di spool SCS mostra come creare i flussi di dati SCS utilizzando la classe SCS3812Writer e come scrivere il flusso su un file di spool su un server
- Esempio: lettura di file di spool mostra come utilizzare PrintObjectInputStream per leggere un file di spool del server esistente
- Esempio: lettura e trasformazione di file di spool mostra come utilizzare PrintObjectPageInputStream e PrintObjectTransformedInputStream per ottenere differenti trasformazioni quando si leggono i dati del file di spool
- Esempio: copia di un file di spool mostra come copiare un file di spool sulla stessa coda che contiene il file che si desidera copiare.
- Esempio: elenco dei file di spool in modalità asincrona (utilizzando i listener) mostra come elencare in modo asincrono tutti i file di spool su un sistema e come utilizzare l'interfaccia PrintObjectListListener per visualizzare il feedback non appena viene creato l'elenco
- Esempio: elenco dei file di spool in modalità asincrona (senza utilizzare i listener) mostra come elencare in modo asincrono tutti i file di spool su un sistema *senza* utilizzare l'interfaccia PrintObjectListListener
- Esempio: elenco dei file di spool in modalità asincrona mostra come elencare in modo asincrono tutti i file di spool su un sistema

Elenco degli oggetti di stampa:

E' possibile utilizzare la classe PrintObjectList e le sue sottoclassi per gestire gli elenchi degli oggetti di stampa. Ogni sottoclasse contiene metodi che consentono di filtrare l'elenco in base a ciò che risulta appropriato per quel particolare tipo di oggetto di stampa. Ad esempio, SpooledFileList consente di filtrare un elenco di file di spool in base all'utente che ha creato i file di spool, alla coda di emissione su cui si trovano i file di spool, il tipo di formato o i dati utente dei file di spool. Vengono elencati solo quei file di spool che corrispondono ai criteri di filtraggio. Se non è stato impostato alcun filtro, viene utilizzato un valore predefinito per ogni filtro.

Per richiamare effettivamente un elenco di oggetti di stampa dal server, vengono utilizzati i metodi openSynchronously() o openAsynchronously(). Il metodo openSynchronously() non viene restituito fino a che tutti gli oggetti nell'elenco non vengono richiamati dal server. Il metodo openAsynchronously() viene restituito immediatamente e il programma di chiamata può effettuare altre operazioni in primo piano mentre attende la creazione dell'elenco. L'elenco aperto in modo asincrono consente anche al programma di chiamata di avviare la visualizzazione degli oggetti all'utente quando gli oggetti vengono restituiti. Poiché l'utente può visualizzare gli oggetti nel momento in cui vengono restituiti, il tempo di risposta gli può sembrare più veloce. In realtà, il tempo di risposta può effettivamente essere, nel complesso, più lungo, a causa dell'elaborazione supplementare effettuata su ogni oggetto nell'elenco.

Se l'elenco viene aperto in modo asincrono, il programma di chiamata può ottenere un feedback sulla creazione dell'elenco. I metodi, come isCompleted() e size(), indicano se la creazione dell'elenco è terminata o restituiscono la dimensione corrente dell'elenco. Altri metodi, waitForListToComplete() e waitForItem(), consentono al programma di chiamata di attendere che l'elenco sia completo oppure di attendere una voce particolare. Inoltre, per richiamare questi metodi PrintObjectList, il programma di chiamata si può registrare con l'elenco come listener. In questa situazione, il programma di chiamata è informato sugli eventi che si verificano nell'elenco. Per registrare o per annullare la registrazione degli eventi, il programma di chiamata utilizza PrintObjectListListener(), quindi richiama addPrintObjectListListener() per la registrazione oppure removePrintObjectListListener() per annullare la registrazione. La seguente tabella mostra gli eventi che sono distribuiti da un PrintObjectList.

Evento PrintObjectList	Quando l'evento è distribuito
listClosed	Quando l'elenco viene chiuso.
listCompleted	Quando l'elenco è completo.
listErrorOccurred	Se viene inviata un'eccezione quando l'elenco viene richiamato.
listOpened	Quando l'elenco è aperto.
listObjectAdded	Quando un oggetto viene aggiunto all'elenco.

Una volta aperto l'elenco e una volta elaborati gli oggetti dell'elenco, chiudere l'elenco utilizzando il metodo close(). In questo modo vengono liberate tutte le risorse assegnate al programma di raccolta dati inutili durante l'operazione di apertura. Una volta chiuso l'elenco, i filtri possono essere modificati e l'elenco stesso può essere riaperto.

Quando vengono elencati gli oggetti di stampa, gli attributi relativi a ogni oggetto di stampa elencato vengono inviati dal server e memorizzati con l'oggetto di stampa. Questi attributi possono essere aggiornati utilizzando il metodo update() nella classe PrintObject. Quali attributi vengono restituiti dal server dipende dal tipo di oggetto di stampa elencato. Esiste un elenco predefinito di attributi per ogni tipo di oggetto di stampa che può essere sostituito utilizzando il metodo setAttributesToRetrieve() in PrintObjectList. Consultare la sezione Richiamo degli attributi PrintObject per un elenco degli attributi supportati da ogni tipo di oggetto di stampa.

L'elenco delle risorse AFP è consentito solo nella Versione 3 Release 7 e successivi release di OS/400. L'apertura di un AFPResourceList su un sistema precedente a V3R7 crea un'eccezione RequestNotSupportedException.

Esempi

Gli esempi che seguono mostrano modi differenti di elencare i file di spool.

"Esempio: elenco dei file di spool in modalità asincrona (utilizzando i listener)" a pagina 519 mostra come elencare in modo asincrono tutti i file di spool su un sistema e come utilizzare l'interfaccia PrintObjectListListener per ottenere feedback mentre viene creato l'elenco

"Esempio: elenco dei file di spool in modalità asincrona (senza utilizzare i listener)" a pagina 523 mostra come elencare in modo asincrono tutti i file di spool su un sistema *senza* utilizzare l'interfaccia PrintObjectListListener

"Esempio: elenco dei file di spool in modalità sincrona" a pagina 524 mostra come elencare in modo sincrono tutti i file di spool su un sistema

Gestione degli oggetti di stampa:

PrintObject è una classe astratta.(Una classe astratta non consente di creare un'istanza della classe. Invece, è necessario creare una istanza di una delle relative sottoclassi.) Creare gli oggetti delle sottoclassi in ognuno dei seguenti modi:

- Se si conosce il sistema e gli attributi di identificazione dell'oggetto, creare esplicitamente l'oggetto richiamando il programma di creazione pubblico.
- E' possibile utilizzare una sottoclasse PrintObjectList per creare un elenco degli oggetti, quindi, richiamare i singoli oggetti tramite l'elenco.
- Un oggetto può essere creato e restituito come risultato della chiamata a un metodo o ad una serie di metodi. Ad esempio, il metodo statico start() nella classe WriterJob restituisce un oggetto WriterJob.

Utilizzare la classe di base, PrintObject e le relative sottoclassi per gestire gli oggetti di stampa del server:

- OutputQueue
- Printer
- PrinterFile
- SpooledFile
- WriterJob

Richiamo degli attributi PrintObject:

E' possibile richiamare gli attributi PrintObject utilizzando l'ID dell'attributo ed uno di questi metodi dalla classe PrintObject di base:

- Utilizzare getIntegerAttribute(int attributeID) per richiamare un attributo di tipo integer.
- UtilizzaregetFloatAttribute(int attributeID) per richiamare un attributo di tipo floating point.
- UtilizzaregetStringAttribute(int attributeID) per richiamare un attributo di tipo string.

Il parametro attributeID è un numero intero che identifica quale attributo richiamare. Tutti gli ID vengono definiti come costanti pubbliche nella classe PrintObject di base. Il file PrintAttributes contiene una voce di ogni ID dell'attributo.La voce include una descrizione dell'attributo ed il tipo (integer, floating point, string). Per un elenco di attributi da richiamare utilizzando questi metodi, selezionare i seguenti collegamenti:

- AFPResourceAttrs per le risorse AFP
- OutputQueueAttrs per le code di emissione
- PrinterAttrs per le stampanti
- PrinterFileAttrs per i file di stampa
- SpooledFileAttrs per i file di spool
- WriterJobAttrs per i lavori del programma di scrittura

Per ottenere prestazioni accettabili, questi attributi vengono copiati sul client. Essi vengono copiati quando vengono elencati gli oggetti o la prima volta in cui essi risultano necessari se l'oggetto è stato creato in modo implicito. Ciò impedisce all'oggetto di spostarsi sull'host ogni volta che l'applicazione deve richiamare un attributo. Consente, inoltre, all'istanza PrintObject Java di contenere informazioni obsolete sull'oggetto sul server. L'utente dell'oggetto può aggiornare tutti gli attributi richiamando il metodo update() sull'oggetto.Inoltre, se l'applicazione richiama un qualsiasi metodo sull'oggetto che potrebbe causare la modifica degli attributi dell'oggetto, gli attributi vengono aggiornati automaticamente. Ad esempio, se una coda di emissione ha un attributo di stato di RELEASED (getStringAttribute(ATTR_OUTQSTS); restituisce una stringa "RELEASED" e il metodo hold() viene richiamato sulla coda di emissione, richiamando l'attributo dello stato dopo che ha restituito HELD.

Metodo setAttributes

E' possibile utilizzare il metodo setAttributes per modificare gli attributi degli oggetti file di spool e file di stampa. Selezionare i seguenti collegamenti per un elenco o per gli attributi da impostare:

- File PrinterFileAttrs per i file di stampa
- SpooledFileAttrs per i file di spool

Il metodo setAttributes richiede un parametro PrintParameterList, che è una classe utilizzata per conservare una collezione di ID di attributi e dei relativi valori. L'elenco in avvio è vuoto ed il chiamante può aggiungere gli attributi all'elenco utilizzando i vari metodi setParameter() su essa.

Classe PrintParameterList

E' possibile utilizzare la classe PrintParameterList per passare un gruppo di attributi ad un metodo che considera un certo numero di attributi come parametri. Ad esempio, è possibile inviare un file di spool

che utilizza TCP (LPR) tramite il metodo SpooledFile, sendTCP().L'oggetto PrintParameterList contiene i parametri richiesti per il comando di invio, ad esempio il sistema remoto e la coda, più tutti i parametri facoltativi desiderati, ad esempio cancellare il file di spool dopo averlo inviato. In questi casi, la documentazione sul metodo fornisce un elenco di attributi obbligatori o facoltativi. Il metodo PrintParameterList setParameter() non verifica quali attributi si stanno impostando ed i valori ad essi assegnati. Il metodo PrintParameterList setParameter() contiene semplicemente i valori da inoltrare insieme al metodo. In generale, attributi supplementari nel PrintParameterList vengono ignorati e valori non validi negli attributi utilizzati sono diagnosticati sul server.

Attributi della risorsa AFP:

Richiamo degli attributi

Gli attributi che seguono possono essere richiamati per una risorsa AFP utilizzando il metodo getIntegerAttribute(), getStringAttribute() o getFloatAttribute() appropriato:

- ATTR_AFP_RESOURCE - Percorso IFS della risorsa AFP
- ATTR_OBJEXTATTR - Attributo esteso dell'oggetto
- ATTR_DESCRIPTION - Descrizione testo
- ATTR_DATE - Data apertura file
- ATTR_TIME - Ora apertura file
- ATTR_NUMBYTES - Numero di byte da leggere/scrivere

Impostazione degli attributi

Non è consentito impostare attributi per la risorsa AFP.

Attributi della coda di emissione:

Come richiamare gli attributi

E' possibile richiamare i seguenti attributi per una coda di emissione utilizzando il metodo getIntegerAttribute(), getStringAttribute() o getFloatAttribute() appropriato:

- ATTR_AUTHCHCK - Autorizzazione al controllo
- ATTR_DATA_QUEUE - Nome IFS della coda dati
- ATTR_DISPLAYANY - Visualizzare tutti i file
- ATTR_JOBSEPRATR - Separatori lavoro
- ATTR_NUMFILES - Numero di file
- ATTR_NUMWRITERS - Numero di programmi di scrittura avviati in coda
- ATTR_OPCNTRL - Operatore controllato
- ATTR_ORDER - Ordine dei file in coda
- ATTR_OUTPUT_QUEUE - Nome IFS della coda di emissione
- ATTR_OUTQSTS - Stato della coda di emissione
- ATTR_PRINTER - Stampante
- ATTR_SEPPAGE - Pagina separatore
- ATTR_DESCRIPTION - Descrizione testo
- ATTR_USRDEFOPT - Opzione/i definite dall'utente
- ATTR_USER_DEFINED_OBJECT - Nome IFS dell'oggetto definito dall'utente
- ATTR_USER_TRANSFORM_PROG - Nome IFS programma conversione utente
- ATTR_USER_DRIVER_PROG - Nome IFS del programma di controllo utente
- ATTR_WTRJOBNAME - Nome lavoro programma di scrittura
- ATTR_WTRJOBNUM - Numero lavoro programma di scrittura
- ATTR_WTRJOBSTS - Stato lavoro programma di scrittura

- ATTR_WTRJOBUSER - Nome utente programma di scrittura

Impostazione degli attributi

Non è consentito impostare gli attributi per una coda di emissione.

Attributi stampante:
Richiamo degli attributi

E' possibile richiamare i seguenti attributi per un file di stampa utilizzando l'appropriato metodo getIntegerAttribute(), getStringAttribute() o getFloatAttribute() :

- ATTR_AFP - Funzione di stampa avanzata
- ATTR_ALIGNFORMS - Allineare moduli
- ATTR_ALWDRTPRT - Consentire stampa diretta
- ATTR_BTWNCPYSTS - Stato tra copie
- ATTR_BTWNFILESTS - Stato tra file
- ATTR_CODEPAGE - Code Page
- ATTR_CHANGES - Modifiche
- ATTR_DEVCLASS - Classe unità
- ATTR_DEVMODEL - Modello unità
- ATTR_DEVTYPE - Tipo unità
- ATTR_DEVSTATUS - Stato unità
- ATTR_DRWRSEP - Cassetto della carta per i separatori
- ATTR_ENDPNDSTS - Fine stato in sospeso
- ATTR_FILESEP - Separatori file
- ATTR_FONTID - Identificativo font
- ATTR_FORM_DEFINITION - Nome IFS definizione modulo
- ATTR_FORMTYPE - Tipo modulo
- ATTR_FORMTYPEMSG - Opzione messaggio tipo modulo
- ATTR_FORMFEED - Avanzamento pagina
- ATTR_CHAR_ID - Serie di caratteri grafici
- ATTR_HELDSTS - Stato congelato
- ATTR_HOLDPNDSTS - Congelare stato in sospeso
- - Utente lavoro
- ATTR_MFGTYPE - Modello e tipo produttore
- ATTR_MESSAGE_QUEUE - Nome IFS coda messaggi
- ATTR_ONJOBQSTS - Stato coda lavori
- ATTR_OUTPUT_QUEUE - Nome IFS coda di emissione
- ATTR_OVERALLSTS - Stato complessivo
- ATTR_POINTSIZE - Dimensione punto
- ATTR_PRINTER - Stampante
- ATTR_PRTDEVTYPE - Tipo unità di stampa
- ATTR_PUBINF_COLOR_SUP - Colore delle informazioni sulla pubblicazione supportato
- ATTR_PUBINF_PPM_COLOR - Pagine delle informazioni sulla pubblicazione al minuto (Colore)
- ATTR_PUBINF_PPM - Pagine delle informazioni sulla pubblicazione al minuto (Monocromatico)
- ATTR_PUBINF_DUPLEX_SUP - Supporto fronte/retro delle informazioni sulla pubblicazione
- ATTR_PUBINF_LOCATION - Ubicazione delle informazioni sulla pubblicazione

- ATTR_RMTLOCNAME - Nome ubicazione remota
- ATTR_SPOOLFILE - Nome file di spool
- ATTR_SPLFNUM - Numero file di spool
- ATTR_STARTEDBY - Avviato dall'utente
- ATTR_DESCRIPTION - Descrizione testo
- ATTR_USERDATA - Dati utente
- ATTR_USRDEFOPT - Opzione/i definita/e dall'utente
- ATTR_USER_DEFINED_OBJECT - Nome IFS dell'oggetto definito dell'utente
- ATTR_USER_TRANSFORM_PROG - Nome IFS del programma di conversione dell'utente
- ATTR_USER_DRIVER_PROG - Nome IFS del programma di controllo dell'utente
- ATTR_SCS2ASCII - Convertire SCS in ASCII
- ATTR_WTNGDATASTS - Attesa stato dati
- ATTR_WTNGDEVSTS - Attesa stato unità
- ATTR_WTNGMSGSTS - Attesa stato messaggio
- ATTR_WTRAUTOEND - Quando chiudere automaticamente il programma di scrittura
- ATTR_WTRJOBNAME - Nome lavoro del programma di scrittura
- ATTR_WTRJOBSTS - Stato lavoro del programma di scrittura
- ATTR_WTRSTRTD - Programma di scrittura avviato
- ATTR_WRTNGSTS - Stato di scrittura

Impostazione degli attributi

Non è consentito impostare attributi per una stampante.

Attributi file di stampa:
Come richiamare gli attributi

E' possibile richiamare i seguenti attributi per un file di stampa utilizzando il metodo appropriato getIntegerAttribute(), getStringAttribute() o getFloatAttribute() :

- ATTR_ALIGN - Allineare pagina
- - Scostamento trasversale margine posteriore
- ATTR_BKMGN_DWN - Scostamento verso il basso margine posteriore
- ATTR_BACK_OVERLAY - Nome IFS sovrapposizione posteriore
- ATTR_BKOVL_DWN - Scostamento verso il basso sovrapposizione posteriore
- ATTR_BKOVL_ACR - Scostamento trasversale sovrapposizione posteriore
- ATTR_CPI - Caratteri per pollice
- ATTR_CODEDFNTLIB - Nome libreria font codificato
- ATTR_CODEPAGE - Code Page
- ATTR_CODEDFNT - Nome font codice
- ATTR_CONTROLCHAR - Carattere di controllo
- ATTR_CONVERT_LINEDATA - Convertire dati riga
- ATTR_COPIES - Copie
- ATTR_CORNER_STAPLE - Graffettatura nell'angolo
- ATTR_DBCSDATA - Dati DBCS specificati dall'utente
- ATTR_DBCSEXTENSN - Caratteri estensione DBCS
- ATTR_DBCSROTATE - Rotazione carattere DBCS
- ATTR_DBCSCPI - Caratteri DBCS per pollice

- ATTR_DBCSSISO - Spaziatura DBCS SO/SI
- ATTR_DFR_WRITE - Rimandare scrittura
- ATTR_PAGRTT - Grado di rotazione della pagina
- ATTR_EDGESTITCH_NUMSTAPLES - Numero di punti graffettatura a bordo foglio
- ATTR_EDGESTITCH_REF - Riferimento graffettatura a bordo foglio
- ATTR_EDGESTITCH_REFOFF - Riferimento graffettatura a bordo foglio
- ATTR_ENDPAGE - Pagina finale
- ATTR_FILESEP - Separatori file
- ATTR_FOLDREC - Record a capo
- ATTR_FONTID - Identificativo font
- ATTR_FORM_DEFINITION - Nome IFS definizione modulo
- ATTR_FORMFEED - Avanzamento pagina
- ATTR_FORMTYPE - Tipo modulo
- ATTR_FTMGN_ACR - Scostamento trasversale margine anteriore
- ATTR_FTMGN_DWN - Scostamento verso il basso margine anteriore
- ATTR_FRONT_OVERLAY - Nome IFS sovrapposizione anteriore
- ATTR_FTOVL_ACR - Scostamento trasversale sovrapposizione anteriore
- ATTR_FTOVL_DWN - Scostamento verso il basso sovrapposizione anteriore
- ATTR_CHAR_ID - Serie di caratteri grafici
- ATTR_JUSTIFY - Giustificazione hardware
- ATTR_HOLD - Congelare file di spool
- ATTR_LPI - Righe per pollice
- ATTR_MAXRCDS - Numero massimo record emissione spool
- ATTR_OUTPTY - Priorità di emissione
- ATTR_OUTPUT_QUEUE - Nome IFS coda di emissione
- ATTR_OVERFLOW - Numero righe in eccedenza
- ATTR_PAGE_DEFINITION - IFS definizione pagina
- ATTR_PAGELEN - Lunghezza pagina
- ATTR_MEASMETHOD - Metodo di misurazione
- ATTR_PAGEWIDTH - Ampiezza pagina
- ATTR_MULTIUP - Pagine per lato
- ATTR_POINTSIZE - Dimensione punto
- ATTR_FIDELITY - Fedeltà stampa
- ATTR_DUPLEX - Stampa su entrambi i lati
- ATTR_PRTQUALITY - Qualità stampa
- ATTR_PRTTEXT - Testo stampa
- ATTR_PRINTER - Stampante
- ATTR_PRTDEVTYPE - Tipo unità di stampa
- ATTR_RPLUNPRT - Sostituire caratteri non stampabili
- ATTR_RPLCHAR - Carattere di sostituzione
- ATTR_SADDLESTITCH_NUMSTAPLES - Numero di punti graffettatura centrale
- ATTR_SADDLESTITCH_REF - Riferimento graffettatura centrale
- ATTR_SAVE -Salvare file di spool
- ATTR_SRCDRWR - Cassetto origine
- ATTR_SPOOL - Effettuare lo spool dei dati

- ATTR_SCHEDULE - Pianificazione emissione di spool
- ATTR_STARTPAGE - Pagina iniziale
- ATTR_DESCRIPTION - Descrizione testo
- ATTR_UNITOFMEAS - Unità di misura
- ATTR_USERDATA - Dati utente
- ATTR_USRDEFDATA - Dati definiti dall'utente
- ATTR_USRDEFOPT - Opzione/i definita/e dall'utente
- ATTR_USER_DEFINED_OBJECT - Nome IFS dell'oggetto definito dell'utente

Impostazione degli attributi

E' possibile utilizzare i seguenti attributi per un file di stampa utilizzando il metodo setAttributes():

- ATTR_ALIGN - Allineare pagina
- - Scostamento trasversale margine posteriore
- ATTR_BKMGN_DWN - Scostamento verso il basso margine posteriore
- ATTR_BACK_OVERLAY - Nome IFS sovrapposizione posteriore
- ATTR_BKOVL_DWN - Scostamento verso il basso sovrapposizione posteriore
- ATTR_BKOVL_ACR - Scostamento trasversale sovrapposizione posteriore
- ATTR_CPI - Caratteri per pollice
- ATTR_CODEDFNTLIB - Nome libreria font codificato
- ATTR_CODEPAGE - Code Page
- ATTR_CODEDFNT - Nome font codice
- ATTR_CONTROLCHAR - Carattere di controllo
- ATTR_CONVERT_LINEDATA - Convertire dati riga
- ATTR_COPIES - Copie
- ATTR_CORNER_STAPLE - Graffettatura nell'angolo
- ATTR_DBCSDATA - Dati DBCS specificati dall'utente
- ATTR_DBCSEXTENSN - Caratteri estensione DBCS
- ATTR_DBCSROTATE - Rotazione carattere DBCS
- ATTR_DBCSCPI - Caratteri DBCS per pollice
- ATTR_DBCSSISO - Spaziatura DBCS SO/SI
- ATTR_DFR_WRITE - Rimandare scrittura
- ATTR_PAGRTT - Grado di rotazione della pagina
- ATTR_EDGESTITCH_NUMSTAPLES - Numero di punti graffettatura a bordo foglio
- ATTR_EDGESTITCH_REF - Riferimento graffettatura a bordo foglio
- ATTR_EDGESTITCH_REFOFF - Riferimento graffettatura a bordo foglio
- ATTR_ENDPAGE - Pagina finale
- ATTR_FILESEP - Separatori file
- ATTR_FOLDREC - Record a capo
- ATTR_FONTID - Identificativo font
- ATTR_FORM_DEFINITION - Nome IFS definizione modulo
- ATTR_FORMFEED - Avanzamento pagina
- ATTR_FORMTYPE - Tipo modulo
- ATTR_FTMGN_ACR - Scostamento trasversale margine anteriore
- ATTR_FTMGN_DWN - Scostamento verso il basso margine anteriore
- ATTR_FRONT_OVERLAY - Nome IFS sovrapposizione anteriore

- ATTR_FTOVL_ACR - Scostamento trasversale sovrapposizione anteriore
- ATTR_FTOVL_DWN - Scostamento verso il basso sovrapposizione anteriore
- ATTR_CHAR_ID - Serie di caratteri grafici
- ATTR_JUSTIFY - Giustificazione hardware
- ATTR_HOLD - Congelare file di spool
- ATTR_LPI - Righe per pollice
- ATTR_MAXRCDS - Numero massimo record emissione spool
- ATTR_OUTPTY - Priorità di emissione
- ATTR_OUTPUT_QUEUE - Nome IFS coda di emissione
- ATTR_OVERFLOW - Numero righe in eccedenza
- ATTR_PAGE_DEFINITION - IFS definizione pagina
- ATTR_PAGELEN - Lunghezza pagina
- ATTR_MEASMETHOD - Metodo di misurazione
- ATTR_PAGEWIDTH - Ampiezza pagina
- ATTR_MULTIUP - Pagine per lato
- ATTR_POINTSIZE - Dimensione punto
- ATTR_FIDELITY - Fedeltà stampa
- ATTR_DUPLEX - Stampa su entrambi i lati
- ATTR_PRTQUALITY - Qualità stampa
- ATTR_PRTTEXT - Testo stampa
- ATTR_PRINTER - Stampante
- ATTR_PRTDEVTYPE - Tipo unità di stampa
- ATTR_RPLUNPRT - Sostituire caratteri non stampabili
- ATTR_RPLCHAR - Carattere di sostituzione
- ATTR_SADDLESTITCH_NUMSTAPLES - Numero di punti graffettatura centrale
- ATTR_SADDLESTITCH_REF - Riferimento graffettatura centrale
- ATTR_SAVE -Salvare file di spool
- ATTR_SRCDRWR - Cassetto origine
- ATTR_SPOOL - Effettuare lo spool dei dati
- ATTR_SCHEDULE - Pianificazione emissione di spool
- ATTR_STARTPAGE - Pagina iniziale
- ATTR_DESCRIPTION - Descrizione testo
- ATTR_UNITOFMEAS - Unità di misura
- ATTR_USERDATA - Dati utente
- ATTR_USRDEFDATA - Dati definiti dall'utente
- ATTR_USRDEFOPT - Opzione/i definita/e dall'utente
- ATTR_USER_DEFINED_OBJECT - Nome IFS dell'oggetto definito dell'utente

Attributi file di spool:

Richiamo degli attributi

E' possibile richiamare gli attributi seguenti utilizzando il metodo appropriato tragetIntegerAttribute(), getStringAttribute() o getFloatAttribute() :

- ATTR_AFP - Funzione di stampa avanzata
- ATTR_ALIGN - Allineare pagina

- ATTR_BKMGN_ACR - Scostamento trasversale sovrapposizione posteriore
- ATTR_BKMGN_DWN - Scostamento verso il basso sovrapposizione posteriore
- ATTR_BACK_OVERLAY - Nome IFS sovrapposizione posteriore
- ATTR_BKOVL_DWN - Scostamento verso il basso sovrapposizione posteriore
- ATTR_BKOVL_ACR - Scostamento trasversale sovrapposizione posteriore
- ATTR_CPI - Caratteri per pollice
- ATTR_CODEDFNTLIB - Nome libreria font codificato
- ATTR_CODEDFNT - Nome font codice
- ATTR_CODEPAGE - Code Page
- ATTR_CONTROLCHAR - Carattere di controllo
- ATTR_COPIES - Copie
- ATTR_COPIESLEFT - Altre copie da produrre
- ATTR_CORNER_STAPLE - Graffettatura nell'angolo
- ATTR_CURPAGE - Pagina corrente
- ATTR_DATE - Data creazione oggetto
- ATTR_DATE_WTR_BEGAN_FILE - Data in cui il programma di scrittura ha avviato l'elaborazione del file di spool
- ATTR_DATE_WTR_CMPL_FILE - Data in cui il programma di scrittura ha completato l'elaborazione del file di spool
- ATTR_DBCSDATA - Dati DBCS specificati dall'utente
- ATTR_DBCSEXTENSN - Caratteri estensione DBCS
- ATTR_DBCSROTATE - Rotazione carattere DBCS
- ATTR_DBCSCPI - Caratteri DBCS per pollice
- ATTR_DBCSSISO - Spaziatura DBCS SO/SI
- ATTR_PAGRTT - Grado di rotazione della pagina
- ATTR_EDGESTITCH_NUMSTAPLES - Numero di punti graffettatura a bordo foglio
- ATTR_EDGESTITCH_REF - Riferimento graffettatura a bordo foglio
- ATTR_EDGESTITCH_REFOFF - Scostamento riferimento graffettatura a bordo foglio
- ATTR_ENDPAGE - Pagina finale
- ATTR_FILESEP - Separatori file
- ATTR_FOLDREC - Record a capo
- ATTR_FONTID - Identificativo font
- ATTR_FORM_DEFINITION - Nome IFS definizione modulo
- ATTR_FORMFEED - Avanzamento pagina
- ATTR_FORMTYPE - Tipo modulo
- ATTR_FTMGN_ACR - Scostamento trasversale margine anteriore
- ATTR_FTMGN_DWN - Scostamento verso il basso margine anteriore
- ATTR_FRONTSIDE_OVERLAY - Nome IFS sovrapposizione anteriore
- ATTR_FTOVL_ACR - Scostamento trasversale sovrapposizione anteriore
- ATTR_FTOVL_DWN - Scostamento verso il basso sovrapposizione anteriore
- ATTR_CHAR_ID - Serie di caratteri grafici
- ATTR_JUSTIFY - Giustificazione hardware
- ATTR_HOLD - Congelare file di spool
- ATTR_IPP_ATTR_CHARSET - Attributi-charset IPP
- ATTR_IPP_JOB_ID - ID lavoro IPP

- ATTR_IPP_JOB_NAME - Nome lavoro IPP
- ATTR_IPP_JOB_NAME_NL - NL nome lavoro IPP
- ATTR_IPP_JOB_ORIGUSER - Utente di origine lavoro IPP
- ATTR_IPP_JOB_ORIGUSER_NL - NL Utente di origine lavoro IPP
- ATTR_IPP_PRINTER_NAME - Nome stampante IPP
- ATTR_JOBNAME - Nome lavoro
- ATTR_JOBNUMBER - Numero lavoro
- - Utente lavoro
- ATTR_JOB_SYSTEM - Sistema lavoro
- ATTR_LASTPAGE - Ultima pagina stampata
- ATTR_LINESPACING - Spaziatura riga
- ATTR_LPI - Righe per pollice
- ATTR_MAXRCDS - Numero massimo record emissione spool
- ATTR_PAGELEN - Lunghezza pagina
- ATTR_PAGEWIDTH - Ampiezza pagina
- ATTR_MEASMETHOD - Metodo di misurazione
- ATTR_NETWORK - Identificativo di rete
- ATTR_NUMBYTES - Numero di byte da leggere/scrivere
- ATTR_OUTPUTBIN - Contenitore di emissione
- ATTR_OUTPTY - Priorità di emissione
- ATTR_OUTPUT_QUEUE - Nome IFS coda di emissione
- ATTR_OVERFLOW - Numero righe in eccedenza
- ATTR_MULTIUP - Pagine per lato
- ATTR_POINTSIZE - Dimensione punto
- ATTR_FIDELITY - Fedeltà stampa
- ATTR_DUPLEX - Stampa su entrambi i lati
- ATTR_PRTQUALITY - Qualità stampa
- ATTR_PRTTEXT - Testo stampa
- ATTR_PRINTER - Stampante
- ATTR_PRTASSIGNED - Stampante assegnata
- ATTR_PRTDEVTYPE - Tipo unità di stampa
- ATTR_PRINTER_FILE - Nome IFS file di stampa
- ATTR_RECLENGTH - Lunghezza record
- ATTR_REDUCE - Ridurre emissione
- ATTR_RPLUNPRT - Sostituire caratteri non stampabili
- ATTR_RPLCHAR - Carattere di sostituzione
- ATTR_RESTART - Riavviare stampa
- ATTR_SADDLESTITCH_NUMSTAPLES - Numero di punti graffettatura centrale
- ATTR_SADDLESTITCH_REF - Riferimento graffettatura centrale
- ATTR_SAVE -Salvare file di spool
- ATTR_SRCDRWR - Cassetto origine
- ATTR_SPOOLFILE - Nome file di spool
- ATTR_SPLFNUM - Numero file di spool
- ATTR_SPLFSTATUS - Stato file di spool
- ATTR_SCHEDULE - Pianificazione emissione di spool

- ATTR_STARTPAGE - Pagina iniziale
- ATTR_SYSTEM - Luogo creazione sistema
- ATTR_TIME - Ora creazione oggetto
- ATTR_TIME_WTR_BEGAN_FILE - Ora in cui il programma di scrittura ha avviato l'elaborazione del file di spool
- ATTR_TIME_WTR_CMPL_FILE - Ora in cui il programma di scrittura ha completato l'elaborazione del file di spool
- ATTR_PAGES - Totale pagine
- ATTR_UNITOFMEAS - Unità di misura
- ATTR_USERCMT - Commento utente
- ATTR_USERDATA - Dati utente
- ATTR_USRDEFDATA - Dati definiti dall'utente
- ATTR_USRDEFFILE - File definito dall'utente
- ATTR_USRDEFOPT - Opzione/i definita/e dall'utente
- ATTR_USER_DEFINED_OBJECT - Nome IFS oggetto definito dall'utente

Impostazione degli attributi

E' necessario impostare i seguenti attributi per un file di spool utilizzando il metodo setAttributes():
- ATTR_ALIGN - Allineare pagina
- ATTR_BACK_OVERLAY - Nome IFS sovrapposizione posteriore
- ATTR_BKOVL_DWN - Scostamento verso il basso sovrapposizione posteriore
- ATTR_BKOVL_ACR - Scostamento trasversale sovrapposizione posteriore
- ATTR_COPIES - Copie
- ATTR_ENDPAGE - Pagina finale
- ATTR_FILESEP - Separatori file
- ATTR_FORM_DEFINITION - Nome IFS definizione modulo
- ATTR_FORMFEED - Avanzamento pagina
- ATTR_FORMTYPE - Tipo modulo
- ATTR_FRONTSIDE_OVERLAY - Nome IFS sovrapposizione anteriore
- ATTR_FTOVL_ACR - Scostamento trasversale sovrapposizione anteriore
- ATTR_FTOVL_DWN - Scostamento verso il basso sovrapposizione anteriore
- ATTR_OUTPTY - Priorità di emissione
- ATTR_OUTPUT_QUEUE - Nome IFS coda di emissione
- ATTR_MULTIUP - Pagine per lato
- ATTR_FIDELITY - Fedeltà stampa
- ATTR_DUPLEX - Stampa su entrambi i lati
- ATTR_PRTQUALITY - Qualità stampa
- ATTR_PRTSEQUENCE - Sequenza di stampa
- ATTR_PRINTER - Stampante
- ATTR_RESTART - Riavviare stampa
- ATTR_SAVE -Salvare file di spool
- ATTR_SCHEDULE - Pianificazione emissione di spool
- ATTR_STARTPAGE - Pagina iniziale
- ATTR_USERDATA - Dati utente
- ATTR_USRDEFOPT - Opzione/i definita/e dall'utente

- ATTR_USER_DEFINED_OBJECT - Nome IFS oggetto definito dall'utente

Attributi lavoro programma di scrittura:

Richiamo degli attributi

E' possibile richiamare un lavoro del programma di scrittura utilizzando il metodo getIntegerAttribute(), getStringAttribute() o getFloatAttribute():

- ATTR_WTRJOBNAME - Nome lavoro del programma di scrittura
- ATTR_WTRJOBNUM - Numero lavoro del programma di scrittura
- ATTR_WTRJOBSTS - Stato lavoro del programma di scrittura
- ATTR_WTRJOBUSER - Nome utente lavoro del programma di scrittura

Impostazione degli attributi

Non è consentito impostare gli attributi per un lavoro del programma di scrittura.

Attributi PrintObject:

Indice analitico

- Funzione di stampa avanzata
- Risorsa AFP
- Allineare moduli
- Allineare pagina
- Consentire stampa diretta
- Autorizzazione
- Autorizzazione al controllo
- Programma di scrittura a chiusura automatica
- Memoria ausiliaria
- Scostamento trasversale margine posteriore
- Scostamento verso il basso margine posteriore
- Sovrapposizione lato posteriore
- Scostamento trasversale sovrapposizione posteriore
- Scostamento verso il basso sovrapposizione posteriore
- Stato tra copie
- Stato tra file
- Modifiche
- Caratteri per pollice
- Code page
- Nome font codice
- Nome libreria font codificato
- Carattere di controllo
- Convertire dati riga
- Copie
- Altre copie da produrre
- Graffettatura nell'angolo
- Pagina corrente
- Formato dati
- Coda dati

- Data apertura file
- Data fine creazione lavoro file di spool
- Data in cui il programma di scrittura ha avviato l'elaborazione del file di spool
- Data in cui il programma di scrittura ha completato l'elaborazione del file di spool
- Dati DBCS specificati dall'utente
- Caratteri estensione DBCS
- Rotazione carattere DBCS
- Caratteri DBCS per pollice
- Spaziatura DBCS SO/SI
- Rimandare scrittura
- Grado di rotazione della pagina
- Cancellare il file dopo l'invio
- Opzione destinazione
- Tipo destinazione
- Classe unità
- Modello unità
- Stato unità
- Tipo unità
- Visualizzare ogni file
- Cassetto della carta per i separatori
- Numero di punti graffettatura a bordo foglio
- Riferimento graffettatura a bordo foglio
- Scostamento riferimento graffettatura a bordo foglio
- Fine stato in sospeso
- Pagina finale
- Origine buste
- Separatori file
- Record a capo
- Identificativo font
- Definizione modulo
- Avanzamento pagina
- Tipo modulo
- Opzione messaggio tipo modulo
- Scostamento trasversale margine anteriore
- Scostamento verso il basso margine anteriore
- Sovrapposizione anteriore
- Scostamento trasversale sovrapposizione anteriore
- Scostamento verso il basso sovrapposizione anteriore
- Serie caratteri grafici
- Giustificazione hardware
- Stato congelato
- Congelare file di spool
- Congelare stato in sospeso
- Configurazione immagine
- Inizializzare il programma di scrittura

- Indirizzo internet
- Attributi-charset IPP
- ID lavoro IPP
- Nome lavoro IPP
- NL nome lavoro IPP
- Nome utente che crea lavoro IPP
- NL nome utente che crea lavoro IPP
- Nome stampante IPP
- Nome lavoro
- Numero lavoro
- Separatori lavoro
- Sistema lavoro
- Utente lavoro
- Ultima pagina stampata
- Lunghezza pagina
- Nome libreria
- Righe per pollice
- Spaziatura riga
- Modello e tipo produttore
- Elenco numero massimo di lavori per client
- Numero massimo record emissione spool
- Metodo di misurazione
- Messaggio aiuto
- ID messaggio
- Coda messaggi
- Risposta messaggio
- Testo messaggio
- Tipo messaggio
- Severità messaggio
- Capacità di risposta a più elementi
- Identificativo di rete
- Attributi oggetto del server stampa di rete
- Numero di byte nel file di spool
- Numero di byte in lettura/scrittura
- Numero di file
- Numero di programmi di scrittura avviati in coda
- Attributo esteso dell'oggetto
- Stato coda lavori
- Aprire comandi ora
- Operatore controllato
- Ordine dei file in coda
- Contenitore di emissione
- Priorità di emissione
- Coda di emissione
- Stato coda di emissione

- Stato complessivo
- Numero righe in eccedenza
- Una pagina alla volta
- Conteggio pagine stimate
- Definizione pagina
- Numero pagina
- Pagine per lato
- Origine carta 1
- Origine carta 2
- Densità pel
- Dimensione punto
- Fedeltà stampa
- Stampa su entrambi i lati
- Qualità stampa
- Sequenza stampa
- Testo stampa
- Stampante
- Stampante assegnata
- Tipo unità di stampa
- File di stampa
- Coda di stampa
- Colore delle informazioni sulla pubblicazione supportato
- Pagine delle informazioni sulla pubblicazione al minuto (colore)
- Pagine delle informazioni sulla pubblicazione al minuto (monocromatico)
- Supporto fronte/retro delle informazioni sulla pubblicazione
- Ubicazione delle informazioni sulla pubblicazione
- Nome ubicazione remota
- Lunghezza record
- Riduzione emissione
- Sistema remoto
- Sostituire caratteri non stampabili
- Carattere di sostituzione
- Riavviare stampa
- Numero graffettatura centrale dei punti
- Riferimento graffettatura centrale
- Salvare file di spool
- Ricerca scostamento
- Ricerca origine
- Priorità di invio
- Pagina separatore
- Cassetto origine
- SCS spool
- Effettuare lo spool dei dati
- Metodo autenticazione creazione del file di spool
- Metodo sicurezza creazione del file di spool

- Nome file di spool
- Numero file di spool
- Stato file di spool
- Pianificazione emissione di spool
- Avviato dall'utente
- Pagina iniziale
- Luogo creazione sistema
- Descrizione testo
- Ora apertura del file
- Ora fine creazione lavori del file di spool
- Ora in cui il programma di scrittura ha avviato l'elaborazione del file di spool
- Ora in cui il programma di scrittura ha completato l'elaborazione del file di spool
- Totale pagine
- Convertire SCS in ASCII
- Unità di misura
- Commento utente
- Dati utente
- Dati definiti dall'utente
- File definito dall'utente
- Oggetto definito dall'utente
- Opzione/i definita/e dall'utente
- Dati programma del programma di controllo utente
- Programma del programma di controllo utente
- ID utente
- Indirizzo ID utente
- Programma conversione utente
- Visualizzazione fedeltà
- Classe VM/MVS
- Attesa stato dati
- Attesa stato unità
- Attesa stato messaggio
- Quando chiudere automaticamente il programma di scrittura
- Quando chiudere il programma di scrittura
- Quando congelare il file
- Ampiezza pagina
- Oggetto personalizzazione stazione di lavoro
- Nome lavoro programma di scrittura
- Numero lavoro programma di scrittura
- Stato lavoro programma di scrittura
- Nome utente lavoro programma di scrittura
- Programma di scrittura avviato
- Pagina iniziale del programma di scrittura
- Stato di scrittura
- NPS CCSID
- Livello NPS

Funzione di stampa avanzata

ID ATTR_AFP

Tipo String

Descrizione
Indica se questo file di spool utilizza risorse esterne AFP nel file di spool. I valori validi sono *YES e *NO.

Risorsa AFP

ID ATTR_AFP_RESOURCE

Tipo String

Descrizione
Il percorso IFS della risorsa esterna AFP (Advanced Function Print). Il formato del percorso IFS è ″/QSYS.LIB/library.LIB/resource.type″ dove *library* è la libreria che contiene la risorsa, *resource* è il nome della risorsa e *type* è il tipo di risorsa. I valori validi per *type* includono ″FNTRSC″, ″FORMDF″, ″OVL″, ″PAGSEG″ e ″PAGDFN″.

Allineare moduli

ID ATTR_ALIGNFORMS

Tipo String

Descrizione
L'ora in cui verrà inviato un messaggio di allineamento moduli. I valori validi sono *WTR, *FILE, *FIRST.

Allineare pagina

ID ATTR_ALIGN

Tipo String

Descrizione
Indica se un messaggio di allineamento moduli viene inviato prima di stampare questo file di spool. I valori validi sono *YES, *NO.

Consentire stampa diretta

ID ATTR_ALWDRTPRT

Tipo String

Descrizione
Indica se il programma di scrittura della stampante consente l'assegnazione della stampante ad un lavoro che stampa direttamente ad una stampante. I valori validi sono *YES, *NO.

Autorizzazione

ID ATTR_AUT

Tipo String

Descrizione
Specifica l'autorizzazione concessa agli utenti privi di una specifica autorizzazione sulla coda di emissione. I valori validi sono *USE, *ALL, *CHANGE, *EXCLUDE, *LIBCRTAUT.

Autorizzazione al controllo

ID ATTR_AUTCHK

Tipo String

Descrizione
Indica quale tipo di autorizzazioni alla coda di emissione consente all'utente di controllare tutti i file sulla coda di emissione. I valori validi sono *OWNER, *DTAAUT.

Programma di scrittura a chiusura automatica

ID ATTR_AUTOEND

Tipo String

Descrizione
Indica se il programma di scrittura deve essere chiuso automaticamente. I valori validi sono *NO, *YES.

Memoria ausiliaria

ID ATTR_AUX_POOL

Tipo Numero intero

Descrizione
Specifica il numero di ASP su cui il file di spool viene memorizzato. I valori possibili sono:

- 1: Sistema ASP
- 2-32: Uno degli ASP utente

Scostamento trasversale margine posteriore

ID ATTR_BACKMGN_ACR

Tipo Mobile

Descrizione
Specifica, per il lato posteriore del foglio di carta, la distanza dal lato sinistro della pagina da cui inizia la stampa. Il valore speciale *FRONTMGN verrà codificato come -1.

Scostamento verso il basso margine posteriore

ID ATTR_BACKMGN_DWN

Tipo Mobile

Descrizione
Specifica, per il lato posteriore del foglio di carta, la distanza dall'inizio pagina da cui inizia la stampa. Il valore speciale *FRONTMGN verrà codificato come -1.

Sovrapposizione lato posteriore

ID ATTR_BACK_OVERLAY

Tipo String

Descrizione
Il percorso IFS della sovrapposizione lato posteriore o di un valore speciale. Se il valore è un percorso IFS avrà il formato "/QSYS.LIB/library.LIB/overlay.OVL" dove *library* è la libreria della risorsa e *overlay* è il nome della sovrapposizione. I valori validi speciali includono *FRONTOVL.

Scostamento trasversale sovrapposizione posteriore

ID ATTR_BKOVL_ACR

Tipo Mobile

Descrizione
Lo scostamento trasversale dal punto di origine dove la sovrapposizione viene stampata.

Scostamento verso il basso sovrapposizione posteriore

ID ATTR_BKOVL_DWN

Tipo Mobile

Descrizione
Lo scostamento verso il basso dal punto in cui la sovrapposizione viene stampata.

Stato tra copie

ID ATTR_BTWNCPYSTS

Tipo String

Descrizione
Se il programma di scrittura è tra copie di un file di spool a copia multipla. I valori restituiti sono *YES o *NO.

Stato tra file

ID ATTR_BTWNFILESTS

Tipo String

Descrizione
Se il programma di scrittura è tra file. I valori restituiti sono *YES o *NO.

Modifiche

ID ATTR_CHANGES

Tipo String

Descrizione
L'ora in cui le modifiche in sospeso diventano operative. I valori validi sono *NORDYF, *FILEEND o uno spazio che non implica alcuna modifica in sospeso al programma di scrittura.

Caratteri per pollice

ID ATTR_CPI

Tipo Mobile

Descrizione
Il numero di caratteri per pollice orizzontale.

Code page

ID ATTR_CODEPAGE

Tipo String

Descrizione
La corrispondenza dei caratteri grafici ai punti codice per questo file di spool. Se il campo della serie di caratteri grafici contiene un valore speciale, è possibile che contenga uno zero (0).

Nome font codice

ID ATTR_CODEDFNT

Tipo String

Descrizione
Il nome del font codificato. Un font codificato è una risorsa AFP composta da una serie di caratteri e una code page. I valori speciali includono *FNTCHRSET.

Nome libreria font codificato

ID ATTR_CODEDFNTLIB

Tipo String

Descrizione
Il nome della libreria che contiene il font codificato. E' possibile che questo campo contenga degli spazi se il nome font codificato ha un valore speciale.

Carattere di controllo

ID ATTR_CONTROLCHAR

Tipo String

Descrizione
Se questo file utilizza il carattere di controllo della stampante American National Standards. I possibili valori sono *NONE se nessun carattere di controllo di stampa viene passato ai dati che vengono stampati o *FCFC il quale indica che il primo carattere di ogni record è un carattere di controllo della stampante American National Standards.

Convertire dati riga

ID ATTR_CONVERT_LINEDATA

Tipo String

Descrizione
Se i dati riga sono convertiti in AFPDS prima di essere scritti nello spool. I possibili valori sono *NO e *YES.

Copie

ID ATTR_COPIES

Tipo Numero intero

Descrizione
Il numero totale di copie da produrre per questo file di spool.

Altre copie da produrre

ID ATTR_COPIESLEFT

Tipo Numero intero

Descrizione
Il numero rimanente di copie da produrre per questo file di spool.

Graffettatura nell'angolo

ID ATTR_CORNER_STAPLE

Tipo String

Descrizione
L'angolo di riferimento da utilizzare per una graffettatura nell'angolo. Una graffettatura è trasportata nel supporto magnetico all'angolo riferimento. I valori validi sono *NONE, *DEVD, *BOTRIGHT, *TOPRIGHT, *TOPLEFT e *BOTLEFT.

Pagina corrente

ID ATTR_CURPAGE

Tipo Numero intero

Descrizione
La pagina corrente che viene scritta dal lavoro del programma di scrittura.

Formato dati

ID ATTR_DATAFORMAT

Tipo String

Descrizione
Formato dati. I valori validi sono *RCDDATA, *ALLDATA.

Coda dati

ID ATTR_DATA_QUEUE

Tipo String

Descrizione
Specifica il percorso IFS della coda dati che viene associata alla coda di emissione o a ″*NONE″ se nessuna coda dati è associata alla coda di emissione. Il formato del percorso IFS è ″/QSYS.LIB/library.LIB/dataqueue.DTAQ″ dove *library* è la libreria contenente la coda dati e *dataqueue* è il nome della coda dati.

Data apertura file

ID ATTR_DATE

Tipo String

Descrizione
Per i file di spool questa è la data in cui è stato aperto il file di spool. Per le risorse AFP questa è la data indicante l'ultima modifica dell'oggetto. La data è codificata in una stringa di caratteri con il seguente formato, C YY MM DD.

Data fine creazione lavoro file di spool

ID ATTR_DATE_END

Tipo Stringa

Descrizione
La data di fine del lavoro che ha creato il file di spool sul sistema. Se il campo Data iniziale creazione file di spool è impostato su *ALL, è necessario che il campo stesso venga impostato su spazi. Se è stata specificata una data per il campo Data iniziale creazione file di spool, è necessario che il campo stesso venga impostato su una data valida. La data deve essere in formato CYYMMDD o essere uno dei seguenti valori speciali:

- *LAST: è necessario restituire tutti i file di spool con una data e un'ora di creazione uguali o superiori alla data iniziale di creazione del file di spool.
- Data: è necessario restituire tutti i file di spool con una data e un'ora di creazione uguali o superiori alla data e all'ora di creazione del file di spool e inferiori o uguali alla data e all'ora finale di creazione del file di spool.

Il formato data CYYMMDD è definito come segue:

- C è il secolo, dove 0 indica gli anni 19xx e 1 indica gli anni 20xx
- YY indica l'anno
- MM indica il mese

- DD indica il giorno

Data in cui il programma di scrittura ha avviato l'elaborazione del file di spool

ID ATTR_DATE_WTR_BEGAN_FILE

Tipo String

Descrizione
Indica la data in cui il programma di scrittura ha iniziato l'elaborazione di questo file di spool. La data è codificata in una stringa di caratteri con il seguente formato, C YY MM DD.

Data in cui il programma di scrittura ha completato l'elaborazione del file di spool

ID ATTR_DATE_WTR_CMPL_FILE

Tipo String

Descrizione
Indica la data in cui il programma di scrittura ha avviato l'elaborazione di chiusura di questo file di spool. La data è codificata in una stringa di caratteri con il seguente formato, C YY MM DD.

Dati DBCS specificati dall'utente

ID ATTR_DBCSDATA

Tipo String

Descrizione
Se il file di spool contiene caratteri DBCS (double-byte character set). I valori validi sono *NO e *YES.

Caratteri estensione DBCS

ID ATTR_DBCSEXTENSN

Tipo String

Descrizione
Se il sistema deve elaborare i caratteri estensione DBCS. I valori validi sono *NO e *YES.

Rotazione carattere DBCS

ID ATTR_DBCAROTATE

Tipo String

Descrizione
Se i caratteri DBCS vengono ruotati di 90 gradi in senso antiorario prima della stampa. I valori validi sono *NO e *YES.

Caratteri DBCS per pollice

ID ATTR_DBCSCPI

Tipo Numero intero

Descrizione
Il numero di caratteri DBCS da stampare per pollice. I valori validi sono -1, -2, 5, 6 e 10. Il valore *CPI è codificato come -1. Il valore *CONDENSED è codificato come -2.

Spaziatura DBCS SO/SI

ID ATTR_DBCSSISO

Tipo String

Descrizione
Determina la presentazione dei caratteri di inizio stringa e fine stringa nella stampa. I valori validi sono *NO, *YES e *RIGHT.

Rimandare scrittura

ID ATTR_DFR_WRITE

Tipo String

Descrizione
Se i dati di stampa sono stati precedentemente congelati nei buffer di sistema

Grado di rotazione della pagina

ID ATTR_PAGRTT

Tipo Numero intero

Descrizione
Il grado di rotazione del testo sulla pagina, a seconda del modo in cui il modulo viene caricato nella stampante. I valori validi sono -1, -2, -3, 0, 90, 180, 270. Il valore *AUTO è codificato come -1, il valore *DEVD è codificato come -2 e il valore *COR è codificato come -3.

Cancellare il file dopo l'invio

ID ATTR_DELETESPLF

Tipo String

Descrizione
Cancellare il file di spool dopo l'invio? I valori validi sono *NO e *YES.

Opzione destinazione

ID ATTR_DESTOPTION

Tipo String

Descrizione
Opzione destinazione. Una stringa di testo che consente all'utente di passare le opzioni al sistema ricevente.

Tipo destinazione

ID ATTR_DESTINATION

Tipo String

Descrizione
Tipo destinazione. I valori validi sono *OTHER, *AS400, *PSF2.

Classe unità

ID ATTR_DEVCLASS

Tipo String

Descrizione
La classe unità.

Modello unità

ID ATTR_DEVMODEL

Tipo String

Descrizione
Il numero modello dell'unità.

Stato unità

ID ATTR_DEVSTATUS

Tipo Numero intero

Descrizione
Lo stato dell'unità di stampa. I valori validi sono 0 (disattivato), 10 (disattivo in sospeso), 20 (attivo in sospeso), 30 (attivato), 40 (collegamento in sospeso), 60 (attivo), 66 (programma di scrittura attivo), 70 (congelato), 75 (spento), 80 (ripristino in sospeso), 90 (ripristino annullato), 100 (non riuscito), 106 (programma di scrittura non riuscito), 110 (in fase di manutenzione), 111 (danneggiato), 112 (vincolato), 113 (sconosciuto).

Tipo unità

ID ATTR_DEVTYPE

Tipo String

Descrizione
Il tipo di unità.

Visualizzare ogni file

ID ATTR_DISPLAYANY

Tipo String

Descrizione
Se gli utenti con autorizzazione a leggere questa coda di emissione possono visualizzino i dati di emissione di qualsiasi file di emissione su questa coda o solo i dati nei propri file. I valori validi sono *YES, *NO, *OWNER.

Cassetto della carta per i separatori

ID ATTR_DRWRSEP

Tipo Numero intero

Descrizione
Identifica il cassetto della carta da cui devono essere prese le pagine di separazione del file e lavoro. I valori validi sono -1, -2, 1, 2, 3. Il valore *FILE è codificato come -1 e il valore *DEVD è codificato come -2.

Numero di punti graffettatura a bordo foglio

ID ATTR_EDGESTITCH_NUMSTAPLES

Tipo Numero intero

Descrizione
Il numero di punti applicati lungo l'asse di fine operazione.

Riferimento graffettatura a bordo foglio

ID ATTR_EDGESTITCH_REF

Tipo String

Descrizione
Dove uno o più punti vengono inseriti sul supporto magnetico lungo l'asse di fine operazione. I valori validi sono *NONE, *DEVD, *BOTTOM, *RIGHT, *TOP e *LEFT.

Scostamento riferimento graffettatura a bordo foglio

ID ATTR_EDGESTITCH_REFOFF

Tipo Mobile

Descrizione

Lo scostamento della graffettatura a bordo foglio dal bordo riferimento verso il centro del supporto magnetico.

Fine stato in sospeso

ID ATTR_ENDPNDSTS

Tipo String

Descrizione

Se un comando ENDWTR (Fine programma di scrittura) è stato emesso per questo programma di scrittura. I valori possibili sono *NO - nessun comando ENDWTR è stato emesso, *IMMED - il programma di scrittura si chiude appena i buffer di emissione sono vuoti, *CTRLD - il programma di scrittura si chiude dopo la stampa della copia corrente del file di spool, *PAGEEND - il programma di scrittura si chiude alla fine della pagina.

Pagina finale

ID ATTR_ENDPAGE

Tipo Numero intero

Descrizione

Il numero di pagina in cui terminare la stampa del file di spool. I valori validi sono 0 o il numero di pagina finale. Il valore *END è codificato come 0.

Origine buste

ID ATTR_ENVLP_SOURCE

Tipo String

Descrizione

La dimensione della busta nell'origine buste. Se questo campo non è specificato o se il valore non è valido, viene utilizzato il valore speciale di *MFRTYPMDL. I valori validi sono *NONE - non esiste un'origine buste, *MFRTYPMDL - viene utilizzata la dimensione della busta suggerita dal tipo e dal modello di produzione, *MONARCH (3.875 x 7.5 pollici), *NUMBER9 (3.875 x 8.875 pollici), *NUMBER10 (4.125 x 9.5 pollici), *B5 (176mm x 250mm), *C5 (162mm x 229mm), *DL (110mm x 220mm).

Separatori file

ID ATTR_FILESEP

Tipo Numero intero

Descrizione

Il numero delle pagine del separatore file poste all'inizio di ogni copia del file di spool. I valori validi sono -1 o il numero dei separatori. Il valore *FILE è codificato come -1.

Record a capo

ID ATTR_FOLDREC

Tipo String

Descrizione
Se i record che superano le dimensioni dei moduli della stampante vengono interrotti e riportati alla riga successiva. I valori validi sono *YES, *NO.

Identificativo font

ID ATTR_FONTID

Tipo String

Descrizione
Il font stampante utilizzato. I valori speciali validi includono *CPI e *DEVD.

Definizione modulo

ID ATTR_FORM_DEFINITION

Tipo String

Descrizione
Il nome del percorso IFS della definizione del modulo o di un valore speciale. Se viene specificato un percorso IFS, il formato è "/QSYS.LIB/library.LIB/formdef.FORMDF" dove *library* è la libreria della definizione del modulo e *formdef* è il nome della definizione del modulo. I valori speciali validi includono *NONE, *INLINE, *INLINED e *DEVD.

Avanzamento pagina

ID ATTR_FORMFEED

Tipo String

Descrizione
La modalità di avanzamento dei moduli alla stampante. I valori validi sono *CONT, *CUT, *AUTOCUT, *DEVD.

Tipo modulo

ID ATTR_FORMTYPE

Tipo String

Descrizione
Il tipo di modulo da caricare sulla stampante per stampare questo file di spool.

Opzione messaggio tipo modulo

ID ATTR_FORMTYPEMSG

Tipo String

Descrizione
Opzione messaggio per l'invio di un messaggio alla coda messaggi del programma di scrittura quando il tipo di modulo corrente è terminato. I valori validi sono *MSG, *NOMSG, *INFOMSG, *INQMSG.

Scostamento trasversale margine anteriore

ID ATTR_FTMGN_ACR

Tipo Mobile

Descrizione
Specifica, per il lato anteriore di un foglio di carta, la distanza dal margine sinistro della pagina da cui inizia la stampa. Il valore speciale *DEVD è codificato come -2.

Scostamento verso il basso margine anteriore

ID ATTR_FTMGN_DWN

Tipo Mobile

Descrizione
Specifica, per il lato anteriore di un foglio di carta, la distanza dal margine superiore del foglio da cui inizia la stampa. Il valore speciale *DEVD è codificato come -2.

Sovrapposizione anteriore

ID ATTR_FRONT_OVERLAY

Tipo String

Descrizione
Il percorso IFS della sovrapposizione anteriore. Il formato del percorso IFST è ″/QSYS.LIB/library.LIB/overlay.OVL″ dove *library* è la libreria della risorsa e *overlay* è il nome della sovrapposizione. La stringa ″*NONE″ viene utilizzata per indicare che nessuna sovrapposizione anteriore è stata specificata.

Scostamento trasversale sovrapposizione anteriore

ID ATTR_FTOVL_ACR

Tipo Mobile

Descrizione
Lo scostamento trasversale dal punto di origine dove la sovrapposizione viene stampata.

Scostamento verso il basso sovrapposizione anteriore

ID ATTR_FTOVL_DWN

Tipo Mobile

Descrizione
Lo scostamento verso il basso dal punto in cui la sovrapposizione viene stampata.

Serie caratteri grafici

ID ATTR_CHAR_ID

Tipo String

Descrizione
La serie di caratteri grafici da utilizzare quando si stampa questo file. I valori speciali includono *DEVD, *SYSVAL e *JOBCCSID.

Giustificazione hardware

ID ATTR_JUSTIFY

Tipo Numero intero

Descrizione
La percentuale relativa alla giustificazione a destra dell'emissione. I valori validi sono 0, 50, 100.

Stato congelato

ID ATTR_HELDSTS

Tipo String

Descrizione
Se il programma di scrittura è stato congelato. I valori validi sono *YES, *NO.

Congelare file di spool

ID ATTR_HOLD

Tipo String

Descrizione
Se il file di spool è stato congelato. I valori validi sono *YES, *NO.

Congelare stato in sospeso

ID ATTR_HOLDPNDSTS

Tipo String

Descrizione
Se un comando HLDWTR (Congelamento programma di scrittura) è stato emesso per questo programma di scrittura. I valori possibili sono *NO - nessun comando HLDWTR è stato emesso, *IMMED - il programma di scrittura è congelato quando i buffer di emissione sono vuoti, *CTRLD - programma di scrittura congelato dopo la stampa della copia corrente del file di spool, *PAGEEND - programma di scrittura congelato alla fine della pagina.

Configurazione immagine

ID ATTR_IMGCFG

Tipo Stringa

Descrizione
I servizi di conversione per diversi formati di immagine e di stampa del flusso di dati.

Inizializzare il programma di scrittura

ID ATTR_WTRINIT

Tipo String

Descrizione
L'utente può specificare quando inizializzare l'unità di stampa. I valori validi sono *WTR, *FIRST, *ALL.

Indirizzo internet

ID ATTR_INTERNETADDR

Tipo String

Descrizione
L'indirizzo internet del sistema ricevente.

Attributi-charset IPP

ID ATTR_IPP_ATTR_CHARSET

Tipo String

Descrizione
Indica il charset (serie carattere codificato e metodo di decodificazione) degli attributi del file di spool specificato IPP.

ID lavoro IPP

ID ATTR_IPP_JOB_ID

Tipo Numero intero

Descrizione
L'ID lavoro IPP relativo alla stampante IPP che ha creato il lavoro.

Nome lavoro IPP

ID ATTR_IPP_ATR_CHARSET

Tipo String

Descrizione
Nome utente di facile utilizzo del lavoro.

NL nome lavoro IPP

ID ATTR_IPP_JOB_NAME_NL

Tipo String

Descrizione
Lingua naturale del nome lavoro.

Nome utente che crea lavoro IPP

ID ATTR_IPP_JOB_ORIGUSER

Tipo String

Descrizione
Identifica l'utente finale che ha inoltrato questo lavoro IPP.

NL nome utente che crea lavoro IPP

ID ATTR_IPP_JOB_ORIGUSER_NL

Tipo String

Descrizione
Identifica la lingua naturale del nome utente che dà origine al lavoro.

Nome stampante IPP

ID ATTR_IPP_PRINTER_NAME

Tipo String

Descrizione
Identifica la stampante IPP che ha creato questo lavoro.

Nome lavoro

ID ATTR_JOBNAME

Tipo String

Descrizione
Il nome del lavoro che ha creato il file di spool.

Numero lavoro

ID ATTR_JOBNUMBER

Tipo String

Descrizione
Il numero del lavoro che ha creato il file di spool.

Separatori lavoro

ID ATTR_JOBSEPRATR

Tipo Numero intero

Descrizione
Il numero dei separatori lavoro da posizionare all'inizio dell'emissione per ogni lavoro che dispone di file di spool su questa coda di emissione. I valori validi sono -2, 0-9. Il valore *MSG è codificato come -2. I separatori lavoro vengono specificati alla creazione della coda di emissione.

Sistema lavoro

ID ATTR_JOBSYSTEM

Tipo Stringa

Descrizione
Il lavoro di sistema che ha creato il file di spool era in esecuzione.

Utente lavoro

ID ATTR_JOBUSER

Tipo String

Descrizione
Il nome dell'utente che ha creato il file di spool.

Ultima pagina stampata

ID ATTR_LASTPAGE

Tipo Numero intero

Descrizione
Il numero dell'ultima pagina stampata è il file se la stampa è stata terminata prima dell'elaborazione completa del lavoro.

Lunghezza pagina

ID ATTR_PAGELEN

Tipo Mobile

Descrizione
La lunghezza di una pagina. Le unità di misurazione sono specificate nell'attributo del metodo di misurazione.

Nome libreria

ID ATTR_LIBRARY

Tipo String

Descrizione
Il nome della libreria.

Righe per pollice

ID ATTR_LPI

Tipo Mobile

Descrizione
Il numero delle righe per pollice verticale nel file di spool.

Spaziatura riga

ID ATTR_LINESPACING

Tipo String

Descrizione
Come i record dati riga di un file sono spaziati nella stampa. Le informazioni vengono restituite solo per i file dei tipi di unità di stampa *LINE e *AFPDSLINE. I valori validi sono *SINGLE, *DOUBLE, *TRIPLE o *CTLCHAR.

Modello e tipo produttore

ID ATTR_MFGTYPE

Tipo String

Descrizione
Specifica il produttore, il tipo e il modello quando si convertono i dati di stampa da SCS a ASCII.

Elenco numero massimo di lavori per client

ID ATTR_MAX_JOBS_PER_CLIENT

Tipo Numero intero

Descrizione
Fornito dal client per indicare la dimensione massima della coda di stampa di limitazione.

Numero massimo record emissione spool

ID ATTR_MAXRECORDS

Tipo Numero intero

Descrizione
Il numero massimo di record consentiti in questo file al momento dell'apertura del file stesso. Il valore *NOMAX è codificato come 0.

Metodo di misurazione

ID ATTR_MEASMETHOD

Tipo String

Descrizione
Il metodo di misurazione utilizzato per la lunghezza della pagina e l'ampiezza degli attributi di pagina. I valori validi sono *ROWCOL, *UOM.

Messaggio aiuto

ID ATTR_MSGHELP

Tipo char(*)

Descrizione
Il messaggio di aiuto, a volte conosciuto come testo di secondo livello, può essere restituito da una richiesta ″richiama messaggio″. Il sistema limita la lunghezza a 3000 caratteri (è necessario che la versione inglese sia inferiore del 30% per consentire la traduzione).

ID messaggio

ID ATTR_MESSAGEID

Tipo String

Descrizione
L'ID messaggio.

Coda messaggi

ID ATTR_MESSAGE_QUEUE

Tipo String

Descrizione
Il percorso IFS della coda messaggi che il programma di scrittura utilizza per messaggi operativi. Il formato del percorso IFS è "/QSYS.LIB/library.LIB/messageque.MSGQ" dove *library* è la libreria contenente la coda messaggi e *messageque* è il nome della coda messaggi.

Risposta messaggio

ID ATTR_MSGREPLY

Tipo Stringa

Descrizione
La risposta messaggio. La stringa di testo che deve essere fornita dal client che risponde ad un messaggio del tipo "interrogazione". In caso di richiamo del messaggio, il valore di attributo viene restituito dal server e contiene la risposta predefinita utilizzabile dal client. Il sistema limita la lunghezza a 132 caratteri. Deve essere a chiusura nulla come conseguenza della lunghezza variabile.

Testo messaggio

ID ATTR_MSGTEXT

Tipo Stringa

Descrizione
Il testo messaggio, a volte conosciuto come testo di primo livello, può essere restituito da una richiesta "richiama messaggio". Il sistema limita la lunghezza a 132 caratteri.

Tipo messaggio

ID ATTR_MSGTYPE

Tipo String

Descrizione
Il tipo messaggio, una codifica EBCDIC a 2 cifre. Due tipi di messaggi indicano se uno dei due può "rispondere" ad un messaggio "richiamato": i messaggi informativi '04' trasmettono le informazioni senza richiedere una risposta (possono invece richiedere un'azione correttiva), i messaggi di interrogazione '05' trasmettono le informazioni e richiedono una risposta.

Severità messaggio

ID ATTR_MSGSEV

Tipo Numero intero

Descrizione
Severità messaggio. I valori variano da 00 a 99. Più alto è il valore, più severa o importante è la condizione.

Capacità di risposta a più elementi

ID ATTR_MULTI_ITEM_REPLY

Tipo String

Descrizione
Quando questo valore di attributo è impostato su *YES dal client, le prestazioni delle operazioni del file di spool dell'elenco possono essere migliorate notevolmente. Il valore predefinito è NO.

Identificativo di rete

ID ATTR_NETWORK

Tipo String

Descrizione
L'identificatore di rete del sistema in cui il file è stato creato.

Numero di byte nel file di spool

ID ATTR_NUMBYTES_SPLF

Tipo Numero intero

Descrizione
Il numero totale di byte disponibili nel flusso o nel file di spool. Il valore indica il numero di byte PRIMA di ogni conversione dati. Al fine di conformare file di dimensioni maggiori di 2**31 - 1 byte, questo valore viene adattato; l'utente deve moltiplicare il valore per 10K per ottenere il numero reale di byte. Questo attributo non è valido per i file di spool visualizzati nella modalità di una pagina alla volta.

Numero di byte in lettura/scrittura

ID ATTR_NUMBYTES

Tipo Numero intero

Descrizione
Il numero di byte in lettura per un'operazione di lettura o il numero di byte in scrittura per un'operazione di scrittura. L'azione oggetto determina come interpretare questo attributo.

Numero di file

ID ATTR_NUMFILES

Tipo Numero intero

Descrizione
Il numero dei file di spool esistenti sulla coda di emissione.

Numero di programmi di scrittura avviati in coda

ID ATTR_NUMWRITERS

Tipo Numero intero

Descrizione
Il numero dei lavori del programma di scrittura avviati nella coda di emissione.

Attributo esteso dell'oggetto

ID ATTR_OBJEXTATTR

Tipo String

Descrizione

Un attributo ″esteso″ utilizzato da alcuni oggetti come le risorse font. Questo valore viene visualizzato tramite i comandi WRKOBJ e DSPOBJD sul server. Su un pannello server il titolo può solo indicare ″Attributo″. Nel caso di tipi oggetto di risorse font, ad esempio, i valori comuni sono CDEPAG, CDEFNT e FNTCHRSET.

Stato coda lavori

ID ATTR_ONJOBQSTS

Tipo String

Descrizione

Se il programma di scrittura è su una coda lavori e se, quindi, non è in esecuzione. I valori possibili sono *YES, *NO.

Aprire comandi ora

ID ATTR_OPENCMDS

Tipo String

Descrizione

Indica se l'utente desidera che SCS apra i comandi ora per essere inserito in un flusso di dati prima dei dati del file di spool. I valori validi sono *YES, *NO.

Operatore controllato

ID ATTR_OPCNTRL

Tipo String

Descrizione

Se agli utenti con autorizzazione al controllo lavoro è consentito gestire o controllare i file di spool su questa coda. I valori validi sono *YES, *NO.

Ordine dei file in coda

ID ATTR_ORDER

Tipo String

Descrizione

L'ordine dei file di spool su questa coda di emissione. I valori validi sono *FIFO, *JOBNBR.

Contenitore di emissione

ID ATTR_OUTPUTBIN

Tipo Numero intero

Descrizione

Il contenitore di emissione che la stampante utilizza per l'emissione stampata. I valori variano da 1 a 65535. Il valore *DEVD è codificato come 0.

Priorità di emissione

ID ATTR_OUTPTY

Tipo String

Descrizione

La priorità del file di spool. La priorità varia da 1 (il più alto) a 9 (il più basso). I valori validi sono 0-9, dove 0 rappresenta *JOB.

Coda emissioni

ID ATTR_OUTPUT_QUEUE

Tipo String

Descrizione
Il percorso IFS della coda di emissione. Il formato del percorso IFS è "/QSYS.LIB/library.LIB/queue.OUTQ" dove *library* è la libreria contenente la coda di emissione e *queue* è il nome della coda di emissione.

Stato coda di emissione

ID ATTR_OUTQSTS

Tipo String

Descrizione
Lo stato della coda di emissione. I valori validi sono RELEASED, HELD.

Stato complessivo

ID ATTR_OVERALLSTS

Tipo Numero intero

Descrizione
Lo stato complessivo della "stampante logica". "Stampante logica" si riferisce all'unità di stampa, alla coda di emissione e al lavoro del programma di scrittura. I valori validi sono 1 (non disponibile), 2 (disattivo o ancora non disponibile), 3 (arrestato), 4 (messaggio in attesa), 5 (congelato), 6 (arresto in sospeso), 7 (congelamento in sospeso), 8 (in attesa della stampante), 9 (in attesa dell'avvio), 10 (in stampa), 11 (in attesa della coda di emissione), 12 (collegamento in sospeso), 13 (disattivo), 14 (inutilizzabile), 15 (in manutenzione), 999 (sconosciuto).

Numero righe in eccedenza

ID ATTR_OVERFLOW

Tipo Numero intero

Descrizione
L'ultima riga da stampare prima che i dati in stampa si riversino nella pagina successiva.

Una pagina alla volta

ID ATTR_PAGE_AT_A_TIME

Tipo String

Descrizione
Specifica se è necessario che il file di spool venga aperto con modalità di una pagina alla volta. I valori validi sono *YES e *NO.

Conteggio pagine stimate

ID ATTR_PAGES_EST

Tipo String

Descrizione
Specifica se il conteggio delle pagine è stimato invece di reale. I valori validi sono *YES e *NO.

Definizione pagina

ID ATTR_PAGE_DEFINITION

Tipo String

Descrizione

Il nome del percorso IFS della definizione della pagina o di un valore speciale. Se viene specificato un percorso IFS il formato è ″/QSYS.LIB/library.LIB/pagedef.PAGDFN″ dove *library* è la libreria della definizione pagina e *pagedef* è il nome della definizione pagina. I valori validi speciali includono *NONE.

Numero pagina

ID ATTR_PAGENUMBER

Tipo Numero intero

Descrizione

Il numero della pagina da leggere da un file di spool aperto in modalità di una pagina alla volta.

Pagine per lato

ID ATTR_MULTIUP

Tipo Numero intero

Descrizione

Il numero delle pagine logiche che vengono stampate su ogni lato di ogni pagina fisica quando il file viene stampato. I valori validi sono 1, 2, 4.

Origine carta 1

ID ATTR_PAPER_SOURCE_1

Tipo String

Descrizione

La dimensione del foglio nell'origine carta uno. Se questo campo non è specificato o se il valore non è valido, viene utilizzato il valore speciale di *MFRTYPMDL. I valori validi sono *NONE - non vi è l'origine carta uno o il foglio viene inserito manualmente nella stampante, *MFRTYPMDL - La dimensione del foglio indicata dal tipo e dal modello di produzione, *LETTER (8.5 x 11.0 pollici), *LEGAL (8.5 x 14.0 pollici), *EXECUTIVE (7.25 x 10.5 pollici), *LEDGER (17.0 x 11.0 pollici), *A3 (297mm x 420mm), *A4 (210mm x 297mm), *A5 (148mm x 210mm), *B4 (257mm x 364mm), *B5 (182mm x 257mm), *CONT80 (largo 8.0 pollici con modulo continuo), *CONT132 (largo 13.2 pollici con modulo continuo).

Origine carta 2

ID ATTR_PAPER_SOURCE_2

Tipo String

Descrizione

La dimensione del foglio nell'origine carta due. Se questo campo non è specificato o se il valore non è valido, viene utilizzato il valore speciale di *MFRTYPMDL. I valori validi sono *NONE - non vi è l'origine carta due o il foglio viene manualmente inserito nella stampante, *MFRTYPMDL - viene utilizzata la dimensione del foglio indicata dal tipo e dal modello di produzione, *LETTER (8.5 x 11.0 pollici), *LEGAL (8.5 x 14.0 pollici), *EXECUTIVE (7.25 x 10.5 pollici), *LEDGER (17.0 x 11.0 pollici), *A3 (297mm x 420mm), *A4 (210mm x 297mm), *A5 (148mm x 210mm), *B4 (257mm x 364mm), *B5 (182mm x 257mm), *CONT80 (largo 8.0 pollici con modulo continuo), *CONT132 (largo 13.2 pollici con modulo continuo).

Densità pel

ID ATTR_PELDENSITY

Tipo String

Descrizione
Solo per le risorse font, questo valore è una codifica del numero di pel (″1″ rappresenta una dimensione pel di 240, ″2″ rappresenta una dimensione pel di 320). Valori aggiuntivi potrebbero divenire significativi mentre il server li definisce.

Dimensione punto

ID ATTR_POINTSIZE

Tipo Mobile

Descrizione
La dimensione punto in cui viene stampato il testo di questo file di spool. Il valore speciale *NONE sarà codificato come 0.

Fedeltà stampa

ID ATTR_FIDELITY

Tipo String

Descrizione
Il tipo di gestione errore eseguito durante la stampa. I valori validi sono *ABSOLUTE, *CONTENT.

Stampa su entrambi i lati

ID ATTR_DUPLEX

Tipo String

Descrizione
Come le informazioni vengono stampate. I valori validi sono *FORMDF, *NO, *YES, *TUMBLE.

Qualità stampa

ID ATTR_PRTQUALITY

Tipo String

Descrizione
La qualità stampa utilizzata quando si stampa questo file di spool. I valori validi sono *STD, *DRAFT, *NLQ, *FASTDRAFT.

Sequenza stampa

ID ATTR_PRTSEQUENCE

Tipo String

Descrizione
Sequenza stampa. I valori validi sono *NEXT.

Testo stampa

ID ATTR_PRTTEXT

Tipo String

Descrizione
Il testo stampato alla fine di ogni pagina dell'emissione stampata e sulle pagine separatore. I valori speciali includono *BLANK e *JOB.

Stampante

ID ATTR_PRINTER

Tipo String

Descrizione
Il nome dell'unità di stampa.

Stampante assegnata

ID ATTR_PRTASSIGNED

Tipo String

Descrizione
Indica se la stampante è assegnata. I valori validi sono 1 (assegnata ad una specifica stampante), 2 (assegnata a più stampanti), 3 (non assegnata).

Tipo unità di stampa

ID ATTR_PRTDEVTYPE

Tipo String

Descrizione
Il tipo flusso di dati della stampante. I valori validi sono *SCS, *IPDS, *USERASCII, *AFPDS, *LINE.

File di stampa

ID ATTR_PRINTER_FILE

Tipo String

Descrizione
Il percorso IFS del file di stampa. Il formato del percorso IFS è "/QSYS.LIB/library.LIB/printerfile.FILE" dove *library* è la libreria contenente il file di stampa e *printerfile* è il nome del file di stampa.

Coda di stampa

ID ATTR_RMTPRTQ

Tipo String

Descrizione
Il nome della coda di stampa destinazione quando si inviano file di spool tramite SNDTCPSPLF (LPR).

Colore delle informazioni sulla pubblicazione supportato

ID ATTR_PUBINF_COLOR_SUP

Tipo Stringa

Descrizione
Indica il colore supportato per questa voce dell'elenco di pubblicazione.

Pagine delle informazioni sulla pubblicazione al minuto (colore)

ID ATTR_PUBINF_PPM_COLOR

Tipo Numero intero

Descrizione
Le pagine al minuto supportate nella modalità colore per questa voce elenco pubblicazione.

Pagine delle informazioni sulla pubblicazione al minuto (monocromatico)

ID ATTR_PUBINF_PPM

Tipo Numero intero

Descrizione
Le pagine al minuto supportate in monocromatico per questa voce elenco di pubblicazione.

Supporto fronte/retro delle informazioni sulla pubblicazione

ID ATTR_PUBINF_DUPLEX_SUP

Tipo Stringa

Descrizione
L'indicatore fronte/retro supportato per questa voce elenco pubblicazione.

Ubicazione delle informazioni sulla pubblicazione

ID ATTR_PUBINF_LOCATION

Tipo Stringa

Descrizione
La descrizione dell'ubicazione per questa voce elenco pubblicazione.

Nome ubicazione remota

ID ATTR_RMTLOCNAME

Tipo Stringa

Descrizione
Il nome dell'ubicazione dell'unità di stampa.

Lunghezza record

ID ATTR_RECLENGTH

Tipo Numero intero

Descrizione
Lunghezza record.

Riduzione emissione

ID ATTR_REDUCE

Tipo String

Descrizione
Il modo in cui più pagine logiche vengono stampate su ogni lato di una pagina fisica. I valori validi sono *TEXT o ????.

Sistema remoto

ID ATTR_RMTSYSTEM

Tipo String

Descrizione
Nome sistema remoto. I valori validi speciali includono *INTNETADR.

Sostituire caratteri non stampabili

ID ATTR_RPLUNPRT

Tipo String

Descrizione
Se i caratteri che è impossibile stampare devono essere sostituiti con un altro carattere. I valori validi sono *YES o *NO.

Carattere di sostituzione

ID ATTR_RPLCHAR

Tipo String

Descrizione
Il carattere che sostituisce ogni carattere non stampabile.

Riavviare stampa

ID ATTR_RESTART

Tipo Numero intero

Descrizione
Riavviare la stampa. I valori validi sono -1, -2, -3 o il numero pagina da cui iniziare. Il valore *STRPAGE è codificato come -1, il valore *ENDPAGE è codificato come -2 e il valore *NEXT è codificato come -3.

Numero graffettatura centrale dei punti

ID ATTR_SADDLESTITCH_NUMSTAPLES

Tipo Numero intero

Descrizione
Il numero dei punti da applicare lungo l'asse di fine operazione.

Riferimento graffettatura centrale

ID ATTR_SADDLESTITCH_REF

Tipo String

Descrizione
Uno o più punti vengono trasportati nel supporto magnetico lungo l'asse di fine operazione, posizionato al centro del supporto magnetico parallelo al margine di riferimento. I valori validi sono *NONE, *DEVD, *TOP, and *LEFT.

Salvare file di spool

ID ATTR_SAVESPLF

Tipo String

Descrizione
Se è necessario salvare il file di spool dopo averlo scritto. I valori validi sono *YES, *NO.

Ricerca scostamento

ID ATTR_SEEKOFF

Tipo Numero intero

Descrizione
Ricerca scostamento. Abilita sia i valori positivi che negativi relativi all'origine.

Ricerca origine

ID ATTR_SEEKORG

Tipo Numero intero

Descrizione
I valori validi includono 1 (inizio o parte superiore), 2 (corrente) e 3 (fine o parte inferiore).

Priorità di invio

ID ATTR_SENDPTY

Tipo String

Descrizione
Priorità di invio. I valori validi sono *NORMAL, *HIGH.

Pagina separatore

ID ATTR_SEPPAGE

Tipo String

Descrizione
Consente o meno l'utente di utilizzare l'opzione di stampa di una pagina di intestazione. I valori validi sono *YES o *NO.

Cassetto origine

ID ATTR_SRCDRWR

Tipo Numero intero

Descrizione
Il cassetto della carta da utilizzare quando viene selezionata l'opzione di alimentazione automatica a fogli singoli. I valori validi sono -1, -2, 1-255. Il valore *E1 è codificato come -1 e il valore *FORMDF è codificato come -2.

SCS spool

ID ATTR_SPLSCS

Tipo Long

Descrizione
Determina come i dati SCS vengono utilizzati durante la creazione del file di spool.

Effettuare lo spool dei dati

ID ATTR_SPOOL

Tipo String

Descrizione
Se si effettua lo spool sui dati di emissione per l'unità di stampa. I valori validi sono *YES, *NO.

Metodo autenticazione creazione del file di spool

ID ATTR_SPLF_AUTH_METHOD

Tipo Numero intero

Descrizione

Indica il metodo autenticazione client utilizzato per creare questo file di spool. I valori validi includono x'00'(*NONE), x'01'(*REQUESTER), x'02'(*BASIC), x'03'(*CERTIFICATE) e 'x'04'(*DIGEST).

Metodo sicurezza creazione del file di spool

ID ATTR_SPLF_SECURITY_METHOD

Tipo String

Descrizione

Indica il metodo sicurezza utilizzato per creare questo file di spool. I valori validi sono x'00'(*NONE), x'01'(*SSL3) e x'02'(*TLS).

Nome file di spool

ID ATTR_SPOOLFILE

Tipo String

Descrizione

Il nome del file di spool.

Numero file di spool

ID ATTR_SPLFNUM

Tipo Numero intero

Descrizione

Il numero del file di spool. I valori speciali consentiti sono -1 e 0. Il valore *LAST è codificato come -1, il valore *ONLY è codificato come 0.

Stato file di spool

ID ATTR_SPLFSTATUS

Tipo String

Descrizione

Lo stato del file di spool. I valori validi sono *CLOSED, *HELD, *MESSAGE, *OPEN, *PENDING, *PRINTER, *READY, *SAVED, *WRITING.

Pianificazione emissione di spool

ID ATTR_SCHEDULE

Tipo String

Descrizione

Specifica, solo per i file di spool, quando il file di spool è disponibile al programma di scrittura. I valori validi sono *IMMED, *FILEEND, *JOBEND.

Avviato dall'utente

ID ATTR_STARTEDBY

Tipo String

Descrizione

Il nome dell'utente che ha avviato il programma di scrittura.

Pagina iniziale

ID ATTR_STARTPAGE

Tipo Numero intero

Descrizione

Il numero di pagina da cui avviare la stampa del file di spool. I valori validi sono -1, 0, 1 o il numero di pagina. Il valore *ENDPAGE è codificato come -1. Per il valore 0, la stampa inizia a pagina 1. Per il valore 1, viene stampato l'intero file.

Luogo creazione sistema

ID ATTR_SYSTEM

Tipo String

Descrizione

Il nome del sistema dove il file di spool è stato creato. Quando è impossibile determinare il nome del sistema dove questo file di spool è stato creato, viene utilizzato il nome del sistema ricevente.

Descrizione testo

ID ATTR_DESCRIPTION

Tipo String

Descrizione

Testo per descrivere un'istanza di un oggetto AS400.

Ora apertura del file

ID ATTR_TIMEOPEN

Tipo String

Descrizione

Per i file di spool questa è l'ora di apertura di questo file di spool. Per le risorse AFP questa è l'ora indicante l'ultima modifica dell'oggetto. L'ora è codificata in una stringa di caratteri con il seguente formato, HH MM SS.

Ora fine creazione lavori del file di spool

ID ATTR_TIME_END

Tipo String

Descrizione

L'ora di fine del lavoro che ha creato il file di spool sul sistema. E' necessario che questo campo sia impostato su spazi quando il valore speciale *ALL viene utilizzato per il campo Data iniziale creazione del file di spool o quando il valore speciale *LAST viene utilizzato per il campo Data finale creazione del file di spool. E' necessario che tale campo abbia un valore impostato se una data viene specificata per il campo Data finale creazione del file di spool. L'ora deve avere il formato HHMMSS, definito come segue:

- HH - Ora
- MM - Minuti
- SS - Secondi

Ora in cui il programma di scrittura ha avviato l'elaborazione del file di spool

ID ATTR_TIME_WTR_BEGAN_FILE

Tipo String

Descrizione
Indica l'ora in cui il programma di scrittura ha avviato l'elaborazione del file di spool. L'ora è codificata in una stringa di caratteri con il seguente formato, HH MM SS.

Ora in cui il programma di scrittura ha completato l'elaborazione del file di spool

ID ATTR_TIME_WTR_CMPL_FILE

Tipo String

Descrizione
Indica l'ora in cui il programma di scrittura ha completato l'elaborazione del file di spool. L'ora è codificata in una stringa di caratteri con il seguente formato, HH MM SS.

Totale pagine

ID ATTR_PAGES

Tipo Numero intero

Descrizione
Il numero di pagine contenute in un file di spool.

Convertire SCS in ASCII

ID ATTR_SCS2ASCII

Tipo String

Descrizione
Se i dati di stampa devono essere convertiti da SCS in ASCII. I valori validi sono *YES, *NO.

Unità di misura

ID ATTR_UNITOFMEAS

Tipo String

Descrizione
L'unità di misura da utilizzare per specificare le distanze. I valori validi sono *CM, *INCH.

Commento utente

ID ATTR_USERCMT

Tipo String

Descrizione
I 100 caratteri di un commento specificato dall'utente che descrive il file di spool.

Dati utente

ID ATTR_USERDATA

Tipo String

Descrizione
I 10 caratteri dei dati specificati dall'utente che descrivono il file di spool. I valori validi speciali includono *SOURCE.

Dati definiti dall'utente

ID ATTR_USRDFNDTA

Tipo String

Descrizione
I dati definiti dall'utente utilizzati dalle applicazioni utente o dai programmi utente specificati, che elaborano i file di spool. Tutti i caratteri sono accettati. La massima dimensione è 255.

File definito dall'utente

ID ATTR_USRDEFFILE

Tipo String

Descrizione
Se il file di spool è stato creato utilizzando un API. I valori validi sono *YES o *NO.

Oggetto definito dall'utente

ID ATTR_USER_DEFINED_OBJECT

Tipo String

Descrizione
Il percorso IFS dell'oggetto definito dall'utente che le applicazioni utente utilizzano per elaborare i file di spool. Il formato del percorso IFS è ″/QSYS.LIB/library.LIB/object.type″ dove *library* è il nome della libreria contenente l'oggetto o uno dei valori speciali %LIBL% o %CURLIB%.*Object* è il nome dell'oggetto e *type* è il tipo oggetto. I valori validi per *type* includono ″DTAARA″, ″DTAQ″, ″FILE″, ″PSFCFG″, ″USRIDX″, ″USRQ″ e ″USRSPC″. La stringa ″*NONE″ viene utilizzata per indicare che non deve essere utilizzato alcun oggetto definito.

Opzione/i definita/e dall'utente

ID ATTR_USEDFNOPTS

Tipo String

Descrizione
Le opzioni definite dall'utente che le applicazioni utente utilizzano per elaborare i file di spool. E' possibile specificare fino a un massimo di 4 opzioni, ogni carattere ha una lunghezza pari a char(10). Tutti i caratteri sono accettati.

Dati programma del programma di controllo utente

ID ATTR_USRDRVPGMDTA

Tipo String

Descrizione
I dati utente da utilizzare con il programma del programma di controllo utente. Tutti i caratteri sono accettati. La dimensione massima è 5000 caratteri.

Programma del programma di controllo utente

ID ATTR_USER_DRIVER_PROG

Tipo String

Descrizione
Il percorso IFS del programma di controllo definito dall'utente che elabora i file di spool. Il formato del percorso IFS è ″/QSYS.LIB/library.LIB/program.PGM″ dove *library* è il nome della libreria contenente il programma e *program* è il nome del programma.La *library* può essere uno dei valori speciali %LIBL% e %CURLIB% o il nome di una libreria specifica. La stringa ″*NONE″ viene utilizzata per indicare che nessun programma di controllo è definito.

ID utente

ID ATTR_TOUSERID

Tipo String

Descrizione
L'ID utente a cui viene inviato il file di spool.

Indirizzo ID utente

ID ATTR_TOADDRESS

Tipo String

Descrizione
L'indirizzo dell'utente a cui viene inviato il file di spool.

Programma conversione utente

ID ATTR_USER_TRANSFORM_PROG

Tipo String

Descrizione
Il percorso IFS del programma di conversione definito dell'utente che converte i dati del file di spool prima di essere elaborati dal programma di controllo. Il formato del percorso IFS è ″/QSYS.LIB/library.LIB/program.PGM″ dove *library* è il nome della libreria contenente il programma e *program* è il nome del programma.La *library* può essere uno dei valori speciali %LIBL% e %CURLIB% o il nome di una libreria specifica. La stringa ″*NONE″ viene utilizzata per indicare che nessun programma di conversione è definito.

Visualizzazione fedeltà

ID ATTR_VIEWING_FIDELITY

Tipo String

Descrizione
L'elaborazione da effettuare quando si visualizza una pagina dei dati del file di spool (in modalità di una pagina alla volta). I valori validi sono *ABSOLUTE e *CONTENT (valore predefinito). Per elaborare i dati privi di mappe di bit (comandi) prima della pagina corrente, si utilizza *ABSOLUTE. Per i file SCS, *CONTENT viene utilizzato per elaborare solo i comandi dell'ora aperti più la pagina corrente. Per i file AFPDS, *CONTENT viene utilizzato per elaborare la prima pagina di dati più la pagina corrente.

Classe VM/MVS

ID ATTR_VMMVSCLASS

Tipo String

Descrizione
Classe VM/MVS. I valori validi sono A-Z e 0-9.

Attesa stato dati

ID ATTR_WTNGDATASTS

Tipo String

Descrizione
Se il programma di scrittura ha scritto tutti i dati attualmente nel file di spool e sta attendendo ulteriori dati. I valori possibili sono *NO - il programma di scrittura non sta attendendo ulteriori dati, *YES - il programma di scrittura ha scritto tutti i dati attualmente nel file di spool e sta attendendo ulteriori dati. Questa condizione si verifica quando il programma di scrittura crea un file di spool di apertura con SCHEDULE(*IMMED) specificato.

Attesa stato unità

ID ATTR_WTNGDEVSTS

Tipo String

Descrizione
Se il programma di scrittura sta aspettando di ottenere l'unità da un lavoro che sta stampando direttamente sulla stampante. I valori sono *NO - il programma di scrittura non sta attendendo l'unità, *YES - il programma di scrittura sta attendendo l'unità.

Attesa stato messaggio

ID ATTR_WTNGMSGSTS

Tipo String

Descrizione
Se il programma di scrittura sta attendendo una risposta a un messaggio di interrogazione. I valori sono *NO e *YES.

Quando chiudere automaticamente il programma di scrittura

ID ATTR_WTRAUTOEND

Tipo String

Descrizione
Quando chiudere il programma di scrittura se deve essere chiuso automaticamente. I valori validi sono *NORDYF, *FILEEND. E' necessario impostare l'attributo chiusura automatica del programma di scrittura su *YES.

Quando chiudere il programma di scrittura

ID ATTR_WTREND

Tipo String

Descrizione
Quando chiudere il programma di scrittura. I valori validi sono *CNTRLD, *IMMED e *PAGEEND. E' differente dalla chiusura automatica del programma di scrittura.

Quando congelare il file

ID ATTR_HOLDTYPE

Tipo String

Descrizione
Quando congelare il file di spool. I valori validi sono *IMMED e *PAGEEND.

Ampiezza pagina

ID ATTR_PAGEWIDTH

Tipo Mobile

Descrizione
L'ampiezza di una pagina. Le unità di misurazione sono specificate nell'attributo del metodo di misurazione.

Oggetto personalizzazione stazione di lavoro

ID ATTR_WORKSTATION_CUST_OBJECT

Tipo String

Descrizione
Il percorso IFS dell'oggetto di personalizzazione della stazione di lavoro. Il formato del percorso IFS è "/QSYS.LIB/library.LIB/custobj.WSCST" dove *library* è la libreria contenente l'oggetto personalizzazione e *custobj* è il nome dell'oggetto personalizzazione della stazione di lavoro.

Nome lavoro programma di scrittura

ID ATTR_WRITER

Tipo String

Descrizione
Il nome del lavoro programma di scrittura.

Numero lavoro programma di scrittura

ID ATTR_WTRJOBNUM

Tipo String

Descrizione
Il numero del lavoro programma di scrittura.

Stato lavoro programma di scrittura

ID ATTR_WTRJOBSTS

Tipo String

Descrizione
Lo stato del lavoro programma di scrittura. I valori validi sono STR, END, JOBQ, HLD, MSGW.

Nome utente lavoro programma di scrittura

ID ATTR_WTRJOBUSER

Tipo String

Descrizione
Il nome dell'utente che ha avviato il programma di scrittura.

Programma di scrittura avviato

ID ATTR_WTRSTRTD

Tipo String

Descrizione
Indica se un programma di scrittura è avviato per questa stampante. I valori sono 1 - un programma di scrittura è avviato, 0 - nessun programma di scrittura è avviato.

Pagina iniziale del programma di scrittura

ID ATTR_WTRSTRPAGE

Tipo Numero intero

Descrizione
Specifica il numero della prima pagina da stampare dal primo file di spool quando viene avviato il lavoro programma di scrittura. Ciò è valido solo se viene specificato anche il nome del file di spool quando il programma di scrittura viene avviato.

Stato di scrittura

ID ATTR_WRTNGSTS

Tipo String

Descrizione
Indica se il programma di scrittura della stampante è nello stato scrittura. I valori sono *YES - il programma di scrittura è nello stato scrittura, *NO - il programma di scrittura non è nello stato scrittura, *FILE - il programma di scrittura attende i separatori del file.

Attributi oggetto del server stampa di rete

NPS CCSID

ID ATTR_NPSCCSID

Tipo Numero intero

Descrizione
CCSID in cui verranno codificate tutte le stringhe, come è previsto dal Server stampa di rete.

Livello NPS

ID ATTR_NPSLEVEL

Descrizione
La versione, il release e il livello di modifica del Server stampa di rete. Questo attributo è una stringa di caratteri codificata come VXRYMY (ad esempio ″V3R1M0″) dove

```
X è in (0..9)
Y è in (0..9,A..Z)
```

Copia dei file di spool:

E' possibile utilizzare il metodo di copia della classe SpooledFile per creare una copia del file di spool che l'oggetto SpooledFile rappresenta. Utilizzando SpooledFile.copy() si effettuano le seguenti azioni:

- Si crea il nuovo file di spool sulla stessa coda di emissione e sullo stesso sistema del file di spool originale
- Si restituisce un riferimento al nuovo file di spool

SpooledFile.copy() è un nuovo metodo disponibile per l'utente solo se si scarica JTOpen 3.2 o successive versioni oppure si applica una correzione OS/400. La soluzione ottimale che viene suggerita è scaricare ed utilizzare JTOpen. Per ulteriori informazioni, consultare quanto segue:

IBM Toolbox per Java e JTOpen: Scaricamenti 

IBM Toolbox per Java e JTOpen: Service Pack 

Il metodo di copia utilizza l'API Creazione file di spool (QSPCRTSP) nell'ambito del lavoro server di stampa di rete per creare una replica esatta del file di spool. Sono necessarie solo una data ed un'ora di creazione univoche per preservare l'identità della copia appena creata del file di spool. Per ulteriori informazioni sull'API QSPCRTSP, consultare le seguenti informazioni:

API Creazione file di spool (QSPCRTSP)

Specificando una coda di emissione come parametro per il metodo di copia si crea la copia del file di spool nella prima posizione sulla coda di emissione specificata. Sia la coda di emissione che il file di spool originale devono trovarsi sullo stesso sistema

Esempio: copia di un file di spool utilizzando SpooledFile.copy()

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per informazioni legali importanti.

Questo esempio illustra come utilizzare SpooledFile.copy() per copiare un file di spool nella stessa coda che contiene il file che si desidera copiare. Quando si desidera instradare il file di spool appena copiato in una coda di emissione specifica, passare la coda di emissione come parametro al metodo di copia:

```
SpooledFile newSplf = new sourceSpooledFile.copy(<outqname>);
```

dove <outqname> è l'oggetto OutputQueue.

```
public static void main(String args[]) {
   // Creare l'oggetto sistema
   AS400 as400 = new AS400(<systemname>,<username>, <password>);
   // Identificare la coda di emissione che contiene il file di spool che si desidera copiare.
   OutputQueue outputQueue =
     new OutputQueue(as400, "/QSYS.LIB/QUSRSYS.LIB/<outqname>.OUTQ");

   // Creare una schiera che contiene tutti gli elementi necessari ad
   // identificare in modo univoco un file di spool sul server iSeries.
   String[][] splfTags = { {
      <spoolfilename>,
      <spoolfilenum>,
      <jobname>,
      <username>,
      <jobnumber>,
      // Si noti che <systemname>,<date> e <time> sono facoltativi.
      // Se non vengono inclusi, eliminare i corrispondenti
      // splfTags[i],[j], dove j ha il valore di 5,6 o 7.
      <systemname>,
      <date>,
      <time>},
   };

   // Stampare le informazioni che identificano il file di spool in System.out
   for ( int i=0; i<splfTags.length; i++) {
      System.out.println("Copying -> " + splfTags[i][0] + ","
                                       + splfTags[i][1] + ","
                                       + splfTags[i][2] + ","
                                       + splfTags[i][3] + ","
                                       + splfTags[i][4] + ","
                                       + splfTags[i][5] + ","
                                       + splfTags[i][6] + ","
                                       + splfTags[i][7] );

   // Creare l'oggetto SpooledFile per il file di spool sorgente.
      SpooledFile sourceSpooledFile =
         new SpooledFile(as400,
                         splfTags[i][0],
                         Integer.parseInt(splfTags[i][1]),
                         splfTags[i][2],
                         splfTags[i][3],
                         splfTags[i][5],
                         splfTags[i][6],
                         splfTags[i][7] );
   }

   // Copiare il file di spool, che crea un nuovo oggetto SpooledFile.
   // Per instradare la copia del file di spool in una specifica coda di emissione,
   // utilizzare il seguente codice:
   // SpooledFile newSplf = new sourceSpooledFile.copy(<outqname>);
   // dove <outqname> è un oggetto OutputQueue. Specificare la coda
   // di emissione nel seguente modo:
   // OutputQueue outputQueue =
   //    new OutputQueue(as400, "/QSYS.LIB/QUSRSYS.LIB/<outqname>.OUTQ");
   try { SpooledFile newSplf = new sourceSpooledFile.copy();
```

```
}

        catch (Exception e) {
}
```

Documentazione di riferimento Javadoc

Per ulteriori informazioni su SpooledFile.copy(), consultare la seguente documentazione di riferimento Javadoc:

Il metodo SpooledFile copy()

Creazione di nuovi file di spool:

E' possibile utilizzare la classe SpooledFileOutputStream per creare nuovi file di spool del server.La classe deriva dalla classe JDK standard java.io.OutputStream; dopo la sua creazione, può essere utilizzata ovunque venga utilizzato OutputStream.

Durante la creazione di un nuovo SpooledFileOutputStream, il programma di chiamata può specificare quanto segue:

- Quale file di stampa utilizzare
- In quale coda di emissione inserire il file di spool
- Un oggetto PrintParameterList che può contenere parametri per sovrascrivere i campi nel file di stampa

Questi parametri sono tutti facoltativi (il programma di chiamata può passare valori null per alcuni o per tutti i parametri). Se non viene specificato un file di stampa, il server di stampa di rete utilizza il file di stampa della stampante di rete predefinita, QPNPSPRTF. Il parametro della coda di emissione si trova in quel file per comodità; è possibile specificarlo anche in PrintParameterList. Se il parametro della coda di emissione viene specificato in entrambe le ubicazioni, il campo PrintParameterList sostituisce il parametro della coda di emissione. Consultare la documentazione del programma di creazione dello SpooledFileOutputStream per un elenco completo degli attributi che possono essere impostati in PrintParameterList per la creazione di nuovi file di spool.

Utilizzare uno dei metodi write() per scrivere i dati nel file di spool.L'oggetto SpooledFileOutputStream memorizza nel buffer i dati e li invia quando il flusso di emissione viene chiuso o quando il buffer è pieno. La memorizzazione in buffer viene eseguita per due motivi:

- Consente l'immissione automatica dei dati (consultare Tipi di flusso di dati nei file di spool) per analizzare un buffer pieno di dati per stabilire il tipo di dati
- Consente al flusso di emissione di funzionare più velocemente perché non viene comunicata al server ogni richiesta di scrittura.

Utilizzare il metodo flush() per forzare i dati da scrivere sul server.

Quando il programma di chiamata termina la scrittura dei dati sul nuovo file di spool, viene richiamato il metodo close() per chiudere il file di spool.Una volta che il file di spool è stato chiuso, non è possibile scrivervi i dati. Richiamando il metodo getSpooledFile() una volta chiuso il file di spool, il programma di chiamata può disporre del riferimento ad un oggetto SpooledFile che rappresenta il file di spool.

Tipi di flusso dati nei file di spool

Utilizzare l'attributo Printer Data Type del file di spool per impostare il tipo di dati da inserire nel file di spool. Se il programma di chiamata non specifica un tipo di dati di stampa, il valore predefinito consiste nell'utilizzare l'immissione automatica dei dati. Questo metodo esamina le prime migliaia di byte di dati del file di spool, determina se si adattano alle architetture del flusso dati SCS SNA o AFPDS (Advanced

Function Printing data stream) e imposta l'attributo in modo appropriato. Se i byte di dati del file di spool non corrispondono all'una o all'altra architettura, i dati vengono contrassegnati come *USERASCII. L'immissione automatica dei dati funziona. Il programma di chiamata generalmente utilizza questo a meno che non si trovi nella situazione specifica in cui l'immissione automatica dei dati non funzioni. In tali casi, il programma di chiamata può impostare l'attributo Printer Data Type su per un valore specifico (ad esempio, *SCS). Se il programma di chiamata desidera utilizzare i dati di stampa che si trovano nel file di stampa, deve utilizzare il valore speciale *PRTF. Se il programma di chiamata sostituisce il tipo di dati predefinito durante la creazione di un file di spool, è necessario prestare attenzione per accertarsi che i dati inseriti nel file di spool corrispondano all'attributo del tipo di dati. L'inserimento di dati non SCS in un file di spool contrassegnato per ricevere dati SCS provoca un messaggio di errore dall'host e la perdita del file di spool.

Generalmente, questo attributo può disporre di tre valori:

- ***SCS** - un flusso di dati di stampa EBCDIC basato sul testo.
- ***AFPDS** (Advanced Function Presentation Data Stream) - un altro flusso di dati supportato sul server.*AFPDS può contenere testo, immagine e grafici e può utilizzare risorse esterne come ad esempio le sovrapposizioni di pagina e le immagini esterne nei segmenti della pagina.
- ***USERASCII** - tutti i dati di stampa non SCS e non AFPDS gestiti dal server semplicemente scorrendoli. Flussi di dati Postscript e HP-PCL sono esempi di flussi di dati che si trovano in un file di spool *USERASCII.

Esempi

I seguenti esempi mostrano i modi in cui è possibile gestire i file di spool. Il primo esempio mostra come creare un file di spool su un server da un flusso di immissione. Il secondo esempio mostra come creare un flusso dati SCS utilizzando la classe SCS3812Writer e come scrivere il flusso in un file di spool sul server.

"Esempio: creazione di file di spool" a pagina 515

"Esempio: creazione di file di spool SCS" a pagina 516

Creazione di un flusso di dati SCS:

Per creare file di spool che verranno stampati su determinate stampanti collegate al server, è necessario creare un flusso di dati SCS (SNA Character Stream). (SCS è un flusso di dati EBCDIC basato sul testo che può essere stampato su stampanti SCS, IPDS o PC.) SCS può essere stampato dopo la conversione tramite l'utilizzo di un emulatore o della conversione stampa host sul server.

E' possibile utilizzare le classi del programma di scrittura SCS per creare tale flusso di dati SCS. Le classi del programma di scrittura SCS convertono i caratteri Unicode di Java e le opzioni di formattazione in un flusso di dati SCS. Cinque classi del programma di scrittura SCS creano livelli variabili di flussi di dati SCS. Il programma di chiamata sceglie il programma di scrittura che corrisponde alla destinazione stampante finale su cui il programma di chiamata o l'utente finale eseguirà la stampa.

Utilizzare le seguenti classi del programma di scrittura SCS per creare un flusso di dati di stampa SCS:

Classe del programma di scrittura SCS	Descrizione
SCS5256Writer	La classe più semplice del programma di scrittura SCS. Supporta il testo, il ritorno a capo, l'alimentazione riga, la nuova riga, l'avanzamento pagina, l'orientamento orizzontale e verticale assoluto, l'orientamento orizzontale e verticale relativo e il formato verticale impostato.

Classe del programma di scrittura SCS	Descrizione
SCS5224Writer	Estende il programma di scrittura 5256 e aggiunge i metodi per impostare CPI (character per inch) e LPI (lines per inch).
SCS5219Writer	Estende il programma di scrittura 5224 e aggiunge il supporto per il margine sinistro, per la sottolineatura, il tipo di formato (foglio o busta), la dimensione del modulo, la qualità di stampa, la code page, la serie di caratteri, il numero del cassetto di origine e il numero del cassetto di destinazione.
SCS5553Writer	Estende il programma di scrittura 5219 ed aggiunge supporto per la rotazione caratteri, le righe della griglia e l'adattamento font. Il 5553 è un flusso di dati DBCS (double byte character set).
SCS3812Writer	Estende il programma di scrittura 5219 e aggiunge il supporto per il grassetto, duplex, l'orientamento del testo ed i font.

Per creare un programma di scrittura SCS, il programma di chiamata necessita di un flusso di emissione e, facoltativamente, una codifica. Il flusso di dati viene scritto nel flusso di emissione. Per creare un file di spool SCS, il programma di chiamata per prima cosa crea un SpooledFileOutputStream e lo utilizza per creare un oggetto del programma di scrittura SCS. Il parametro di codifica fornisce un CCSID dell'EBCDIC di destinazione in cui convertire i caratteri.

Una volta creato il programma di scrittura, utilizzare i metodi write() per il testo di emissione.Utilizzare i metodi carriageReturn(), lineFeed() e newLine() per posizionare il cursore di scrittura sulla pagina.Utilizzare il metodo endPage() per terminare la pagina corrente e iniziare una nuova pagina.

Quando tutti i dati sono stati scritti, utilizzare il metodo close() per terminare il flusso di dati e chiudere il flusso di emissione.

Esempio

Il seguente esempio mostra come creare un flusso di dati SCS utilizzando la classe SCS3812Writer e come scrivere il flusso in un file di spool sul server:

Esempio: creazione di file di spool SCS

Lettura dei file di spool e delle risorse AFP:

E' possibile utilizzare la classePrintObjectInputStream per leggere il contenuto non elaborato di un file di spool o di una risorsa AFP (Advanced Function Printing) dal server.La classe estende la classe JDK standard java.io.InputStream in modo che possa essere utilizzata ovunque viene utilizzato un InputStream.

Ottenere un oggetto PrintObjectInputStream richiamando il metodo getInputStream() su una istanza della classe SpooledFile o il metodo getInputStream() su una istanza della classe AFPResource.Il richiamo di un flusso di immissioni per un file di spool è supportato per la Versione 3 Release 2 (V3R2), V3R7 e versioni successive di OS/400. Il richiamo di flussi di immissioni per le risorse AFP è supportato per la V3R7 e versioni successive.

Utilizzare uno dei metodi read() per leggere dal flusso di immissione.Questi metodi restituiscono tutti il numero di byte effettivamente letti, o -1 se non è stato letto alcun byte ed è stata raggiunta la fine del file.

Utilizzare il metodo available() di PrintObjectInputStream per restituire il numero totale di byte nel file di spool o nella risorsa AFP.La classe PrintObjectInputStream supporta l'indicazione del flusso di immissione, così PrintObjectInputStream restituisce sempre true dal metodo markSupported().Il programma di chiamata può utilizzare i metodi mark() e reset() per spostare la posizione attuale di lettura indietro nel flusso di immissioni.Utilizzare il metodoskip() per spostare la posizione attuale di lettura in avanti nel flusso di immissioni senza leggere i dati.

Esempio

Il seguente esempio mostra come utilizzare PrintObjectInputStream per leggere un file di spool server esistente

Esempio: lettura di file spool

Lettura dei file di spool utilizzando PrintobjectPageInputStream e PrintObjectTransformedInputStream:

E' possibile utilizzare la classe PrintObjectPageInputStream per leggere da un server AFP e da un file di spool SCS una pagina alla volta.

E' possibile ottenere un oggetto PrintObjectPageInputStream con il metodo getPageInputStream().

Utilizzare uno dei metodi read() per leggere dal flusso di immissione.Tutti questi metodi restituiscono il numero di byte effettivamente letti o -1 se non è stato letto alcun byte ed è stata raggiunta la fine della pagina.

Utilizzare il metodo available() di PrintObjectPageInputStream per restituire il numero complessivo di byte nella pagina corrente.La classe PrintObjectPageInputStream supporta il contrassegno del flusso di immissione, così PrintObjectPageInputStream restituisce sempre true dal metodo markSupported().Il programma di chiamata può utilizzare i metodi mark() e reset() per spostare indietro la posizione di lettura corrente nel flusso di immissione in modo tale che le letture successive rileggano gli stessi byte.Il programma di chiamata può utilizzare il metodo skip() per spostare la posizione di lettura in avanti nel flusso di immissione senza leggere i dati.

Tuttavia, quando si desidera convertire un intero flusso di dati del file di spool, utilizzare la classe PrintObjectTransformedInputStream.

Esempio

Il seguente esempio mostra come utilizzare PrintObjectPageInputStream e PrintObjectTransformedInputStream per ottenere differenti trasformazioni quando si leggono i dati del file di spool:

"Esempio: lettura e trasformazione dei file di spool" a pagina 518

Productlicense

La classe ProductLicense consente di richiedere le licenze per i prodotti installati su iSeries. Per essere compatibile con gli altri utenti con licenza iSeries, la classe funziona tramite il supporto product licence iSeries quando si richiede o si rilascia una licenza.

La classe non rinforza la normativa relativa alla licenza ma restituisce un numero sufficiente di informazioni tale che l'applicazione possa rinforzare la normativa. Quando si richiede una licenza, la classe ProductLicense restituirà lo stato della richiesta -- licenza accordata o rifiutata. Se la richiesta viene rifiutata, è necessario che l'applicazione disabiliti la funzionalità che ha richiesto la licenza poiché IBM Toolbox per Java non riconosce la funzionalità da disabilitare.

Utilizzare la classe ProductLicense con il supporto licenza iSeries per rinforzare la licenza della propria applicazione:

- Il lato server della propria applicazione registra i termini di prodotto e di licenza con il supporto licenza iSeries.
- Il lato client della propria applicazione utilizza l'oggetto ProductLicense per richiedere e rilasciare le licenze.

Esempio: scenario ProductLicense

Ad esempio, supponiamo che il cliente acquisti 15 licenze di utilizzo simultaneo per il proprio prodotto. Utilizzo simultaneo sta ad indicare che 15 utenti possono utilizzare il prodotto contemporaneamente, ma non è necessario che siano 15 utenti specifici. Possono essere 15 utenti qualsiasi all'interno dell'organizzazione. Queste informazioni vengono registrate con il supporto licenza iSeries. Mentre gli utenti effettuano il collegamento, la propria applicazione utilizza la classe ProductLicense per richiedere una licenza.

- Quando il numero degli utenti simultanei è inferiore a 15, la richiesta ha esito positivo e si esegue l'applicazione.
- Quando il sedicesimo cliente si collega, la richiesta ProductLicense ha esito negativo. L'applicazione visualizza un messaggio di errore e si chiude.

Quando un utente interrompe l'esecuzione dell'applicazione, l'applicazione rilascia la licenza attraverso la classe ProductLicense. La licenza è ora disponibile per l'utilizzo da parte di un altro utente.

Per ulteriori informazioni e per un esempio di codice, fare riferimento a Javadoc ProductLicense.

Classe ProgramCall

La classe ProgramCall consente al programma Java di richiamare un programma iSeries.E' possibile utilizzare la classe ProgramParameter per specificare i parametri di immissione, di emissione e di immissione/emissione. Se il programma viene eseguito, i parametri di emissione e di immissione/emissione contengono i dati restituiti dal programma iSeries. Se il programma iSeries non riesce ad essere eseguito con esito positivo, il programma Java può richiamare ogni messaggio di iSeries risultante come un elenco di oggetti AS400Message .

I parametri richiesti sono i seguenti:

- Il programma ed i parametri da eseguire
- L'oggetto AS400 che rappresenta il server iSeries su cui è installato il programma.

Il nome programma e l'elenco di parametri possono essere impostati sul programma di creazione, tramite il metodo setProgram() o le chiamate sul metodo run(). Il metodo run() chiama il programma.

L'utilizzo della classe ProgramCall consente il collegamento dell'oggetto AS400 al server iSeries. Consultare la gestione dei collegamenti per informazioni sulla gestione dei collegamenti.

Esempio: utilizzo di ProgramCall

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per informazioni legali importanti.

Il seguente esempio mostra come utilizzare la classe ProgramCall:

```
                    // Creare un oggetto AS400.
AS400 sys = new AS400("mySystem.myCompany.com");

                    // Creare un oggetto program. Si desidera eseguire
                    // il programma successivamente.
ProgramCall pgm = new ProgramCall(sys);
```

```
                    // Impostare il nome del programma.
                    // poiché il programma non dispone di
                    // parametri, inoltrare null per
                    // l'argomento ProgramParameter[].
    pgm.setProgram(QSYSObjectPathName.toPath("MYLIB", "MYPROG", "PGM"));

                    // Eseguire il programma.
Il programma non
                    // dispone di parametri. Se l'esecuzione non riesce, l'errore
                    // viene restituito come una serie di messaggi
                    // nell'elenco di messaggi.
    if (pgm.run() != true)
    {
                    // Se si è giunti a questo punto, l'esecuzione del
                    // ha avuto esito negativo. Richiamare l'elenco di
                    // messaggi per determinare il motivo per
                    // cui il programma non è stato eseguito.
       AS400Message[] messageList = pgm.getMessageList();

                    // ... Elaborare l'elenco di messaggi.
    }

                    // Scollegarsi in quanto è terminata
                    // l'esecuzione dei programmi
    sys.disconnectService(AS400.COMMAND);
```

L'oggetto ProgramCall richiede il nome percorso IFS del programma.

Il funzionamento predefinito per i programmi iSeries consiste nell'esecuzione di un lavoro server separato, anche quando il programma Java ed il programma iSeries si trovano sullo stesso server. E' possibile sostituire il funzionamento predefinito e consentire al programma iSeries di operare in un lavoro Java utilizzando il metodo setThreadSafe() .

Utilizzo degli oggetti ProgramParameter

E' possibile utilizzare gli oggetti ProgramParameter per passare dati di parametro tra il programma Java ed il programma iSeries.Impostare i dati di immissione con il metodo setInputData().Dopo l'esecuzione del programma, richiamare i dati di emissione con il metodo getOutputData().Ogni parametro consiste in una schiera di byte. Il programma Java deve convertire la schiera di byte tra i formati Java e iSeries. Le classi di conversione dati forniscono metodi per la conversione dei dati.I parametri vengono aggiunti all'oggetto ProgramCall come un elenco.

Esempio: utilizzo di ProgramParameter

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per informazioni legali importanti.

Il seguente esempio mostra come utilizzare l'oggetto ProgramParameter per passare dati di parametro.

```
                    // Creare un oggetto AS400
    AS400 sys = new AS400("mySystem.myCompany.com");

                    // Il programma dispone di due parametri.
                    // Creare un elenco per conservare questi
                    // parametri.
    ProgramParameter[] parmList = new ProgramParameter[2];

                    // Il primo parametro è un parametro di
                    // immissione
    byte[] key = {1, 2, 3};
    parmList[0] = new ProgramParameter(key);

                    // Il secondo parametro è un parametro di
                    // emissione.
```

```
Viene restituito un numero a
                    // quattro byte.
     parmList[1] = new ProgramParameter(4);

                    // Creare un oggetto programma
                    // specificando il nome del
                    // programma e l'elenco di parametri.
     ProgramCall pgm = new ProgramCall(sys, "/QSYS.LIB/MYLIB.LIB/MYPROG.PGM", parmList);

                    // Eseguire il programma.
     if (pgm.run() != true)
     {

                    // Se iSeries non può eseguire il programma,
                    // consultare l'elenco di messaggi per
                    // individuare il motivo per cui non è stato eseguito.
        AS400Message[] messageList = pgm.getMessageList();

     }
     else
     {
                    // In un altro conteso il programma è stato eseguito. Elaborare
                    // il secondo parametro, che contiene i
                    // dati restituiti.

                    // Creare un programma di conversione per questo
                    // tipo di dati iSeries
        AS400Bin4 bin4Converter = new AS400Bin4();

                    // Convertire dal tipo iSeries all'oggetto Java.
                    // AS400TreePane.
Il numero inizia
                    // all'inizio del buffer.
        byte[] data = parmList[1].getOutputData();
        int i = bin4Converter.toInt(data);
     }

                    // Scollegarsi in quanto è terminata
                    // l'esecuzione dei programmi
     sys.disconnectService(AS400.COMMAND);
```

Classe QSYSObjectPathName

E' possibile utilizzare la classe QSYSObjectPathName per rappresentare un oggetto nell'integrated file system.Utilizzare questa classe per creare un nome dell'integrated file system o per analizzare un nome dell'integrated file system nei suoi componenti.

Diverse classi dell'IBM Toolbox per Java richiedono un nome percorso IFS per poter essere utilizzate. Utilizzare un oggetto QSYSObjectPathName per la creazione del nome.

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

I seguenti esempi mostrano le modalità di utilizzo della classe QSYSObjectPathName:

Esempio 1: l'oggetto ProgramCall richiede il nome IFS del programma server da chiamare. Per la creazione del nome viene utilizzato un oggetto QSYSObjectPathName. Per richiamare il programma PRINT_IT contenuto nella libreria REPORTS utilizzando un QSYSObjectPathName:

```
                    // Creare un oggetto AS400.
     AS400 sys = new AS400("mySystem.myCompany.com");

                    // Creare un oggetto chiamata al programma.
     ProgramCall pgm = new ProgramCall(sys);

                    // Creare un oggetto nome percorso che
```

```
                    // rappresenta il programma PRINT_IT
                    // nella libreria REPORTS.
    QSYSObjectPathName pgmName = new QSYSObjectPathName("REPORTS",
                                                        "PRINT_IT",
                                                        "PGM");

                    // Utilizzare l'oggetto nome percorso per impostare
                    // il nome dell'oggetto chiamata al
                    // AS400TreePane.
    pgm.setProgram(pgmName.getPath());

                    // ... eseguire il programma, elaborare i
                    // risultati
```

Esempio 2: se il nome dell'oggetto AS400 viene utilizzato una sola volta, il programma Java può utilizzare il metodo toPath() per la creazione del nome. Tale metodo è più efficiente rispetto alla creazione di un oggetto QSYSObjectPathName.

```
                    // Creare un oggetto AS400.
    AS400 sys = new AS400("mySystem.myCompany.com");

                    // Creare un oggetto chiamata al programma.
    ProgramCall pgm = new ProgramCall(sys);

                    // Utilizzare il metodo toPath per creare
                    // il nome che rappresenta il programma
                    // PRINT_IT nella libreria REPORTS.
    pgm.setProgram(QSYSObjectPathName.toPath("REPORTS",
                                             "PRINT_IT",
                                             "PGM"));

                    // ... eseguire il programma, elaborare i
                    // risultati
```

Esempio 3: in questo esempio, ad un programma Java viene assegnato un percorso IFS (integrated file system). E' possibile utilizzare la classe QSYSObjectPathName per analizzare questo nome nei suoi componenti:

```
                    // Creare un oggetto nome percorso dal
                    // nome IFS (integrated file system)
                    // completo.
    QSYSObjectPathName ifsName = new QSYSObjectPathName(pathName);

                    // Utilizzare l'oggetto nome percorso per richiamare
                    // la libreria, il nome e il tipo
                    // di oggetto server.
        String library = ifsName.getLibraryName();
        String name    = ifsName.getObjectName();
        String type    = ifsName.getObjectType();
```

Accesso al livello record

Le classi di accesso a livello record consentono di poter effettuare quanto segue:

- Creare un file fisico iSeries specificando uno dei seguenti elementi:
  - La lunghezza del record
  - Un file sorgente DDS (data description specifications) esistente
  - Un oggetto RecordFormat
- Richiamare il formato record da un file fisico o logico iSeries o i formati record da un file logico a formato multiplo iSeries.

  Nota: il formato record del file non è richiamato nella sua interezza. I formati record richiamati sono destinati ad essere utilizzati quando si imposta il formato record per un oggetto AS400File. Vengono

richiamate solo le informazioni sufficienti a descrivere il contenuto di un record del file. Le informazioni sul formato record, come le intestazioni di colonna e gli alias, non vengono richiamate.
- Accedere ai record in un file fisico iSeries in modo sequenziale, per numero record o chiave.
- Scrivere i record su un file iSeries.
- Aggiornare i record in un file iSeries in modo sequenziale, per numero record o chiave.
- Cancellare i record in un file iSeries in modo sequenziale, per numero record o chiave.
- Bloccare un file iSeries per tipi diversi di accesso.
- Utilizzare il controllo di sincronizzazione per consentire ad un programma Java di effettuare le seguenti operazioni:
  - Avviare il controllo di sincronizzazione per il collegamento.
  - Specificare diversi livelli di blocco del controllo sincronizzazione per file differenti.
  - Commit e rollback delle transazioni.
- Cancellare i file iSeries.
- Cancellare un membro da un file iSeries.

Nota: le classi di accesso al livello record non supportano i file logici di unione o i campi chiave null.

Le seguenti classi svolgono le funzioni qui indicate:
- La classe AS400File è la classe base astratta per le classi di accesso a livello record.Fornisce i metodi per l'accesso record sequenziale, per la creazione e la cancellazione dei file, dei membri e per le attività di controllo sincronizzazione.
- La classe KeyedFile rappresenta un file iSeries a cui si accede tramite chiave.
- La classe SequentialFile rappresenta un file iSeries a cui si accede tramite numero record.
- La classe AS400FileRecordDescription fornisce i metodi per richiamare il formato record di un file iSeries.

Le classi di accesso al livello record richiedono un oggetto AS400 che rappresenti il sistema che dispone dei file database.Utilizzando le classi di accesso al livello record, l'oggetto AS400 si collega a iSeries. Consultare la gestione dei collegamenti per informazioni sulla gestione dei collegamenti.

Le classi di accesso al livello record richiedono il nome percorso IFS del file di database. Per ulteriori informazioni, consultare i nomi percorso IFS (Integrated file system).

Le classi di accesso a livello record utilizzano le seguenti classi:
- La classe RecordFormat per descrivere un record del file database
- La classe Record per fornire l'accesso ai record del file database
- La classe LineDataRecordWriter per scrivere un record in formato dati riga

Queste classi sono descritte nella sezione conversione dati.

Esempi
- L'esempio di accesso sequenziale mostra come accedere ad un file iSeries in modo sequenziale.
- L'esempio di lettura file mostra come utilizzare le classi di accesso al livello record per leggere un file iSeries.
- L'esempio file con chiave mostra come utilizzare le classi di accesso a livello record per leggere i record tramite chiave da un file iSeries.

AS400File:

La classe AS400File fornisce i metodi per:
- Creare e cancellare i file fisici e i membri del server

- Leggere e scrivere i record nei file del server
- Vincolare i file per diversi tipi di accesso
- Utilizzare il blocco del record per migliorare le prestazioni
- Impostare la posizione del cursore all'interno di un file del server aperto
- Gestire attività di controllo sincronizzazione

KeyedFile:

La classe KeyedFile fornisce al programma Java accesso ad un file sul server.Accesso con chiave significa che il programma java può accedere ai record di un file specificando una chiave. Esistono dei metodi per posizionare il cursore, leggere, aggiornare e cancellare record tramite chiave.

Per posizionare il cursore utilizzare i seguenti metodi:
- positionCursor(Object[]) - impostare il cursore sul primo record con la chiave specificata.
- positionCursorAfter(Object[]) - impostare il cursore dopo il primo record con la chiave specificata.
- positionCursorBefore(Object[]) - impostare il cursore sul record prima del primo record con la chiave specificata.

Per cancellare un record, utilizzare il metodo seguente:
- deleteRecord(Object[]) - cancellare il primo record con la chiave specificata.

I metodi di lettura sono:
- read(Object[]) - leggere il primo record con la chiave specificata
- readAfter(Object[]) - leggere il record dopo il primo record con la chiave specificata
- readBefore(Object[]) - leggere il record prima del primo record con la chiave specificata
- readNextEqual() - leggere il record successivo la cui chiave corrisponde alla chiave specificata.La ricerca inizia dal record dopo la posizione corrente del cursore.
- readPreviousEqual() - leggere il record precedente la cui chiave corrisponde alla chiave specificata.La ricerca inizia dal record prima della posizione corrente del cursore.

Per aggiornare un record, utilizzare il seguente metodo:
- update(Object[]) - aggiornare il record con la chiave specificata.

Sono forniti anche metodi per specificare un criterio di ricerca quando si posiziona, legge e aggiorna tramite una chiave. Seguono valori di criteri di ricerca validi:
- Uguale - trovare il primo record la cui chiave corrisponde alla chiave specificata.
- Minore di - trovare l'ultimo record la cui chiave precede la chiave specificata nell'ordine delle chiavi del file.
- Minore o uguale - trovare il primo record la cui chiave corrisponde alla chiave specificata.Se nessun record corrisponde alla chiave specificata, trovare l'ultimo record la cui chiave precede la chiave specificata nell'ordine delle chiavi del file.
- Maggiore di - trovare il primo record la cui chiave si trova dopo la chiave specificata nell'ordine delle chiavi.
- Maggiore o uguale - trovare il primo record la cui chiave corrisponde alla chiave specificata.Se i record non corrispondono alla chiave specificata, trovare il primo record la cui chiave si trova dopo la chiave specificata nell'ordine delle chiavi del file.

KeyedFile è una sottoclasse di AS400File; tutti i metodi in AS400File sono disponibili per KeyedFile.

Specifica della chiave

La chiave per un oggetto KeyedFile viene rappresentata da una schiera di Oggetti Java i cui tipi e ordine corrispondono ai tipi e all'ordine dei campi chiave specificati dall'oggetto RecordFormat per il file.

L'esempio che segue mostra come specificare la chiave per l'oggetto KeyedFile.

```
                        // Specificare la chiave per un file i cui campi chiave, nell'ordine,
                        // sono:
                        //    CUSTNAME    CHAR(10)
                        //    CUSTNUM     BINARY(9)
                        //    CUSTADDR    CHAR(100)VARLEN()
                        // Notare che l'ultimo campo è un campo a lunghezza variabile.
     Object[] theKey = new Object[3];
theKey[0] = "John Doe";
     theKey[1] = new Integer(445123);
theKey[2] = "2227 John Doe Lane, ANYTOWN, NY 11199";
```

Un oggetto KeyedFile accetta chiavi parziali e chiavi complete. Tuttavia, i valori del campo chiave che sono specificati devono essere in ordine.

Ad esempio:

```
                        // Specificare una chiave parziale per un file i cui campi chiave,
                        // nell'ordine, sono:
                        //    CUSTNAME    CHAR(10)
                        //    CUSTNUM     BINARY(9)
                        //    CUSTADDR    CHAR(100)VARLEN()
     Object[] partialKey = new Object[2];
partialKey[0] = "John Doe";
     partialKey[1] = new Integer(445123);

                        // Esempio di una chiave parziale INVALID
     Object[] INVALIDPartialKey = new Object[2];
     INVALIDPartialKey[0] = new Integer(445123);
INVALIDPartialKey[1] = "2227 John Doe Lane, ANYTOWN, NY 11199";
```

Le chiavi di valore null e i campi chiave di valore null non sono supportati.

I valori del campo chiave per un record possono essere ottenuti dall'oggetto Record per un file tramite il metodo getKeyFields().

L'esempio che segue mostra come leggere il contenuto di un file grazie alla chiave:

```
                        // Creare un oggetto AS400, il file esiste su questo
                        // server.
AS400 sys = new AS400("mySystem.myCompany.com");

                        // Creare un oggetto file che rappresenta il file
KeyedFile myFile = new KeyedFile(sys, "/QSYS.LIB/MYLIB.LIB/MYFILE.FILE/%FILE%.MBR");
                        // Presupporre che la classe AS400FileRecordDescription sia
                        // stata usata per generare il codice di una sottoclasse di
                        // RecordFormat che rappresenta il formato record
        // del file MYFILE nella libreria MYLIB.  Il codice è stato
                        // compilato ed è disponibile per essere usato dal programma Java.
     RecordFormat recordFormat = new MYKEYEDFILEFormat();

        // Impostare il formato record per myFile.  E' necessario
        // farlo prima di richiamare open()
     myFile.setRecordFormat(recordFormat);

                        // Aprire il file.
     myFile.open(AS400File.READ_WRITE, 0, AS400File.COMMIT_LOCK_LEVEL_NONE);

                        // Il formato record per il file contiene quattro
                        // campi chiave, CUSTNUM, CUSTNAME, PARTNUM
                        // e ORDNUM nell'ordine.
                        // partialKey conterrà 2 valori campo chiave.
```

```
                    // Poiché i valori campo chiave devono essere ordinati,
        // partialKey sarà composto dai valori relativi a
                    // CUSTNUM e CUSTNAME.
     Object[] partialKey = new Object[2];
     partialKey[0] = new Integer(1);
partialKey[1] = "John Doe";

                    // Leggere il primo record corrispondente a partialKey
     Record keyedRecord = myFile.read(partialKey);

                    // Se il record non è stato trovato, verrà restituito null.
     if (keyedRecord != null)
     { // Trovato il record per John Doe, stampare le informazioni.
  System.out.println("Information for customer " + (String)partialKey[1] + ":");
         System.out.println(keyedRecord);
}

                ....

                    // Chiudere il file in quanto è terminato l'utilizzo
     myFile.close();

                   // Scollegarsi in quanto è terminato l'uso dell'accesso a livello record
     sys.disconnectService(AS400.RECORDACCESS);
```

SequentialFile:

La classe SequentialFile fornisce al programma Java accesso ad un file sul server attraverso il numero record.Esistono metodi per posizionare il cursore, leggere, aggiornare e cancellare record in base al numero record.

Per posizionare il cursore utilizzare i seguenti metodi:

- positionCursor(int) - impostare il cursore sul record con il numero record specificato.
- positionCursorAfter(int) - impostare il cursore sul record successivo al numero record specificato.
- positionCursorBefore(int) - impostare il cursore sul record precedente al numero di record specificato.

Per cancellare un record, utilizzare il seguente metodo:

- deleteRecord(int) - cancellare il record con il numero record specificato.

Per leggere un record, utilizzare i seguenti metodi:

- read(int) - leggere il record con il numero record specificato.
- readAfter(int) - leggere il record successivo al numero record specificato.
- readBefore(int) - leggere il record precedente al numero record specificato.

Per aggiornare un record, utilizzare il seguente metodo:

- update(int) - aggiornare il record con il numero record specificato.

SequentialFile è una sottoclasse di AS400File; tutti i metodi in AS400File sono disponibili in SequentialFile.

L'esempio che segue mostra come utilizzare la classe SequentialFile:

```
                                // Creare un oggetto AS400, il file esiste su questo
                                // server.
AS400 sys = new AS400("mySystem.myCompany.com");

                                // Creare un oggetto file che rappresenta il file
     SequentialFile myFile = new SequentialFile(sys, "/QSYS.LIB/MYLIB.LIB/MYFILE.FILE/%FILE%.MBR");

                                // Presupporre che la classe AS400FileRecordDescription sia
```

```
                            // stata usata per generare il codice di una sottoclasse di
                            // RecordFormat che rappresenta il formato record
              // del file MYFILE nella libreria MYLIB.  Il codice è stato
                                    // compilato ed è disponibile per essere usato dal programma Java.
RecordFormat recordFormat = new MYFILEFormat();

              // Impostare il formato record per myFile.  E' necessario
              // farlo prima di richiamare open()
myFile.setRecordFormat(recordFormat);

                            // Aprire il file.
myFile.open(AS400File.READ_WRITE, 0, AS400File.COMMIT_LOCK_LEVEL_NONE);

                                    // Cancellare il record numero 2.
myFile.delete(2);

                                    // Leggere il record numero 5 e aggiornarlo
Record updateRec = myFile.read(5);
updateRec.setField("CUSTNAME", newName);

                                    // Utilizzare il metodo update() della classe di base in quanto
                                    // si è già posizionati sul record.
myFile.update(updateRec);

                                    // Aggiornare il record numero 7
updateRec.setField("CUSTNAME", nextNewName);
updateRec.setField("CUSTNUM", new Integer(7));
myFile.update(7, updateRec);

           ....

                            // Chiudere il file in quanto è terminato l'utilizzo
myFile.close();

                           // Scollegarsi in quanto è terminato l'uso dell'accesso a livello record
sys.disconnectService(AS400.RECORDACCESS);
```

AS400FileRecordDescription:

La classe AS400FileRecordDescription fornisce i metodi per richiamare il formato record di un file sul server.Questa classe fornisce i metodi per la creazione del codice sorgente Java per le sottoclassi di RecordFormat e per la restituzione di oggetti RecordFormat, che descrivono i formati record dei file fisici o logici sul server specificati dall'utente.L'emissione di questi metodi può essere utilizzata come immissione per un oggetto AS400File quando si imposta il formato record.

Si consiglia di utilizzare sempre la classe AS400FileRecordDescription per creare l'oggetto RecordFormat quando il file si trova già sul server.

Nota: la classe AS400FileRecordDescription non richiama l'intero formato record di un file. Vengono richiamate solo le informazioni sufficienti per descrivere il contenuto dei record che costituiscono il file. Informazioni quali l'intestazione di colonna, gli alias e i campi di riferimento non vengono richiamate. Perciò, i formati record richiamati non possono essere necessariamente utilizzati per creare un file il cui formato record sia identico al file da cui è stato richiamato il formato.

Creazione del codice sorgente Java per sottoclassi di RecordFormat per rappresentare il formato record di file sul server

Il metodo createRecordFormatSource() crea file sorgente Java per le sottoclassi della classe RecordFormat.I file possono essere compilati e utilizzati da un'applicazione o da un'applet come immissione nel metodo AS400File.setRecordFormat().

Il metodo createRecordFormatSource() dovrebbe essere utilizzato come strumento di controllo dello sviluppo per richiamare i formati record di file esistenti sul server. Questo metodo consente che il sorgente relativo alla sottoclasse della classe RecordFormat sia creato una sola volta, modificato se necessario, compilato, quindi utilizzato da diversi programmi Java che hanno accesso agli stessi file sul server. Dal momento che questo metodo crea file sul sistema locale, può essere utilizzato solo da applicazioni Java. L'emissione (il codice sorgente Java), tuttavia, può essere compilata e poi utilizzata in modo simile dalle applicazioni e dalle applet Java.

Nota: questo metodo sostituisce i file con lo stesso nome dei file sorgente Java che si stanno creando.

Esempio 1: l'esempio che segue mostra come utilizzare il metodo createRecordFormatSource():

```
                    // Creare un oggetto AS400, il file esiste su questo
                    // server.
AS400 sys = new AS400("mySystem.myCompany.com");

                    // Creare un oggetto AS400FileRecordDescription che rappresenta il file
AS400FileRecordDescription myFile = new AS400FileRecordDescription(sys, "/QSYS.LIB/MYLIB.LIB/MYFILE.FILE");
                    // Creare il file sorgente Java nell'indirizzario di lavoro corrente.
                    // Specificare "package com.myCompany.myProduct;" per l'istruzione
                    // pacchetto nel sorgente che fornirà la classe come
                    // parte del prodotto.
    myFile.createRecordFormatSource(null, "com.myCompany.myProduct");

                    // Presupponendo che il nome formato per il file MYFILE sia FILE1, il
                    // file FILE1Format.java verrà creato nell'indirizzario di lavoro corrente.
                    // Sovrascriverà qualsiasi file con lo stesso nome. Il nome della classe
                    // sarà FILE1Format. La classe verrà estesa da RecordFormat.
```

Esempio 2: compilare il file appena creato, FILE1Format.java e utilizzarlo come segue:

```
                    // Creare un oggetto AS400, il file esiste su questo
                    // server.
AS400 sys = new AS400("mySystem.myCompany.com");

                    // Creare un oggetto AS400File che rappresenti il file
    SequentialFile myFile = new SequentialFile(sys, "/QSYS.LIB/MYLIB.LIB/MYFILE.FILE");

                    // Impostare il formato record
                    // Ciò presuppone che import.com.myCompany.myProduct.FILE1Format;
                    // sia stato eseguito.

    myFile.setRecordFormat(new FILE1Format());

                    // Aprire il file e leggerlo
                    ....

                    // Chiudere il file in quanto è terminato l'utilizzo
    myFile.close();

                    // Scollegarsi in quanto è terminato l'uso dell'accesso a livello record
    sys.disconnectService(AS400.RECORDACCESS);
```

Creazione di oggetti RecordFormat per rappresentare il formato record dei file sul server

Il metodo retrieveRecordFormat() restituisce una schiera di oggetti RecordFormat che rappresentano i formati record di un file esistente sul server.Solitamente, viene restituito solo un oggetto RecordFormat nella schiera. Quando il file per il quale si sta richiamando il formato record è un file logico a formato multiplo, viene restituito più di un oggetto RecordFormat. Utilizzare questo metodo per richiamare dinamicamente il formato record di un file esistente sul server durante il tempo di esecuzione. L'oggetto RecordFormat può essere utilizzato come immissione per il metodo AS400File.setRecordFormat().

L'esempio che segue mostra come utilizzare il metodo retrieveRecordFormat():

```
                    // Creare un oggetto AS400, il file esiste su questo
                    // server.
     AS400 sys = new AS400("mySystem.myCompany.com");

                    // Creare un oggetto AS400FileRecordDescription che rappresenta il file
    AS400FileRecordDescription myFile = new AS400FileRecordDescription(sys, "/QSYS.LIB/MYLIB.LIB/MYFILE.FILE");
                    // Reperire il formato record per il file
          RecordFormat[] format = myFile.retrieveRecordFormat();

                    // Creare un oggetto AS400File che rappresenti il file
          SequentialFile myFile = new SequentialFile(sys, "/QSYS.LIB/MYLIB.LIB/MYFILE.FILE");

                    // Impostare il formato record
          myFile.setRecordFormat(format[0]);

                    // Aprire il file e leggerlo
                    ....

                    // Chiudere il file in quanto è terminato l'utilizzo
          myFile.close();

                    // Scollegarsi in quanto è terminato l'uso dell'accesso a livello record
          sys.disconnectService(AS400.RECORDACCESS);
```

Creazione e cancellazione di file e membri:

I file fisici sul server vengono creati specificando una lunghezza record, un file sorgente DDS del server esistente o un oggetto RecordFormat.

Quando si crea un file e si specifica una lunghezza record, è possibile creare un file dati o un file sorgente. Il metodo imposta il formato record per l'oggetto. Non richiamare il metodo setRecordFormat() per l'oggetto.

Un file dati dispone di un campo. Il nome del campo è il nome del file, il tipo di campo è di tipo carattere e la lunghezza del campo è la lunghezza specificata sul metodo Create.

Un file sorgente dispone di tre campi:

- Il campo SRCSEQ è ZONED DECIMAL (6,2)
- Il campo SRCDAT è ZONED DECIMAL (6,0)
- SRCDTA è un campo di carattere con una lunghezza che è quella specificata sul metodo Create meno 12

Gli esempi che seguono mostrano come creare file e membri.

Esempio 1: per creare un file dati con un record a 128 byte:

```
                                        // Creare un oggetto AS400, il file verrà
                                        // creato su questo server.
     AS400 sys = new AS400("mySystem.myCompany.com");

                                  // Creare un oggetto file che rappresenta il file
          SequentialFile newFile = new SequentialFile(sys, "/QSYS.LIB/MYLIB.LIB/MYFILE.FILE/%FILE%.MBR");

                                        // Creare il file
          newFile.create(128, "*DATA", "Data file with a 128 byte record");

                                        // Aprire il file solo per scriverci.
                                        // Nota: il formato record per il file
                                        // è già stato impostato da create()
          newFile.open(AS400File.WRITE_ONLY, 0, AS400File.COMMIT_LOCK_LEVEL_NONE);

                    // Scrivere un record nel file.  Poiché il record
                                        // è stato impostato su create(), getRecordFormat()
```

```
                                        // può essere richiamato per ottenere un record formattato
                                        // correttamente per questo file.
        Record writeRec = newFile.getRecordFormat().getNewRecord();
        writeRec.setField(0, "Record one");
        newFile.write(writeRec);

                  ....

                                  // Chiudere il file in quanto è terminato l'utilizzo
        newFile.close();
                                        // Scollegarsi poiché è terminato l'utilizzo
                                        // dell'accesso a livello record
        sys.disconnectService(AS400.RECORDACCESS);
```

Esempio 2: quando si crea un file specificando un file sorgente DDS esistente, il file sorgente DDS viene specificato sul metodo create().Il formato record per il file deve essere impostato utilizzando il metodo setRecordFormat() prima di poter aprire il file.Ad esempio:

```
                                        // Creare un oggetto AS400, il file
                                        // verrà creato su questo server.
    AS400 sys = new AS400("mySystem.myCompany.com");

                                        // Creare gli oggetti QSYSObjectPathName sia per
                                        // il nuovo file che per il file DDS.
        QSYSObjectPathName file    = new QSYSObjectPathName("MYLIB", "MYFILE", "FILE", "MBR");
        QSYSObjectPathName ddsFile = new QSYSObjectPathName("MYLIB", "DDSFILE", "FILE", "MBR");

                                  // Creare un oggetto file che rappresenta il file
        SequentialFile newFile = new SequentialFile(sys, file);

                                        // Creare il file
        newFile.create(ddsFile,  "File created using DDSFile description");

                                        // Impostare il formato record per il file
                                        // reperendolo dal server.
        newFile.setRecordFormat(new AS400FileRecordDescription(sys,
        newFile.getPath()).retrieveRecordFormat()[0]);

                                        // Aprire il file per la scrittura
        newFile.open(AS400File.WRITE_ONLY, 0, AS400File.COMMIT_LOCK_LEVEL_NONE);

                     // Scrivere un record nel file.  Il metodo getRecordFormat()
                     // seguito dal metodo getNewRecord() viene usato per richiamare
                     // un record predefinito per il file.
        Record writeRec = newFile.getRecordFormat().getNewRecord();
        newFile.write(writeRec);

                  ....

                                  // Chiudere il file in quanto è terminato l'utilizzo
        newFile.close();
                                        // Scollegarsi poiché è terminato l'utilizzo
                                        // dell'accesso a livello record
        sys.disconnectService(AS400.RECORDACCESS);
```

Esempio 3: quando si crea un file specificando un oggetto RecordFormat, l'oggetto RecordFormat viene specificato sul metodo create().Il metodo imposta il formato record per l'oggetto. Il metodo setRecordFormat() non deve essere chiamato per l'oggetto.

```
                                        // Creare un oggetto AS400, il file verrà creato
                                        // su questo server.
    AS400 sys = new AS400("mySystem.myCompany.com");

                                  // Creare un oggetto file che rappresenta il file
        SequentialFile newFile = new SequentialFile(sys, "/QSYS.LIB/MYLIB.LIB/MYFILE.FILE/%FILE%.MBR");

                     // Reperire il formato record da un file esistente
```

```
RecordFormat recordFormat = new AS400FileRecordDescription(sys,
"/QSYS.LIB/MYLIB.LIB/EXISTING.FILE/MBR1.MBR").retrieveRecordFormat()[0];

                                    // Creare il file
newFile.create(recordFormat,  "File created using record format object");

                                    // Aprire il file solo per scriverci.
                                    // Nota: il formato record per il file
                                    // è già stato impostato da create()
newFile.open(AS400File.WRITE_ONLY, 0, AS400File.COMMIT_LOCK_LEVEL_NONE);

             // Scrivere un record nel file.  L'oggetto recordFormat
                                      // viene utilizzato per richiamare un record predefinito
                                     // formattato in modo appropriato per il file.
Record writeRec = recordFormat.getNewRecord();
newFile.write(writeRec);

           ....

                            // Chiudere il file in quanto è terminato l'utilizzo
newFile.close();
                                    // Scollegarsi poiché è terminato l'utilizzo
                                    // dell'accesso a livello record
sys.disconnectService(AS400.RECORDACCESS);
```

Quando si cancellano i file e i membri, utilizzare questi metodi:

- Utilizzare il metodo delete() per cancellare i file server e tutti i relativi membri.
- Utilizzare il metodo deleteMember() per cancellare solo un membro di un file.

Utilizzare il metodo addPhysicalFileMember() per aggiungere membri ad un file.

Lettura e scrittura di record:

E' possibile utilizzare la classe AS400File per leggere, scrivere aggiornare e cancellare record nei file sul server.Si può accedere al record attraverso la classe Record, descritta da una classe RecordFormat.Il formato record deve essere impostato attraverso il metodo setRecordFormat() prima che il file venga aperto, a meno che il file non fosse stato appena creato (senza la concomitanza di un metodo close()) da uno dei metodi create(), che impostano il formato record per l'oggetto.

Utilizzare i metodi read() per leggere un record dal file. Vengono forniti metodi per effettuare le seguenti operazioni:

- read() - leggere il record che si trova nella posizione corrente del cursore
- readFirst() - leggere il primo record del file
- readLast() - leggere l'ultimo record del file
- readNext() - leggere il record successivo nel file
- readPrevious() - leggere il record precedente nel file

L'esempio che segue mostra come utilizzare il metodo readNext():

```
                            // Creare un oggetto AS400, il file esiste su questo
                            // server.
AS400 sys = new AS400("mySystem.myCompany.com");

                            // Creare un oggetto file che rappresenta il file
SequentialFile myFile = new SequentialFile(sys, "/QSYS.LIB/MYLIB.LIB/MYFILE.FILE/%FILE%.MBR");

                            // Presupporre che la classe AS400FileRecordDescription sia
                            // stata usata per generare il codice di una sottoclasse di
                            // RecordFormat che rappresenta il formato record
                            // del file MYFILE nella libreria MYLIB.  Il codice è stato
                            // compilato ed è disponibile per essere usato dal programma
```

```
                              // Java.
        RecordFormat recordFormat = new MYFILEFormat();

                    // Impostare il formato record per myFile.  E' necessario
                    // farlo prima di richiamare open()
        myFile.setRecordFormat(recordFormat);

                              // Aprire il file.
        myFile.open(AS400File.READ_ONLY, 0, AS400File.COMMIT_LOCK_LEVEL_NONE);

                              // Leggere ogni record nel campo di scrittura del file
                              // CUSTNAME to System.out
        System.out.println("             CUSTOMER LIST");
        System.out.println("_____________________________________________");

        Record record = myFile.readNext();
           while (record != null)
{
            System.out.println(record.getField("CUSTNAME"));
            record = myFile.readNext();
}

                    ....

                              // Chiudere il file in quanto è terminato l'utilizzo
        myFile.close();

                              // Scollegarsi poiché è terminato l'utilizzo
                              // dell'accesso a livello record.
        sys.disconnectService(AS400.RECORDACCESS);
```

Utilizzare il metodo update() per aggiornare il record che si trova nella posizione del cursore.

Ad esempio:

```
          // Creare un oggetto AS400, il file esiste su questo
                               // server.
    AS400 sys = new AS400("mySystem.myCompany.com");

                              // Creare un oggetto file che rappresenta il file
        SequentialFile myFile = new SequentialFile(sys, "/QSYS.LIB/MYLIB.LIB/MYFILE.FILE/%FILE%.MBR");

                              // Presupporre che la classe AS400FileRecordDescription sia
                              // stata usata per generare il codice di una sottoclasse di
                              // RecordFormat che rappresenta il formato record
                              // del file MYFILE nella libreria MYLIB.  Il codice è stato
                              // compilato ed è disponibile per essere usato dal programma Java.
        RecordFormat recordFormat = new MYFILEFormat();

                    // Impostare il formato record per myFile.  E' necessario
                    // farlo prima di richiamare open()
        myFile.setRecordFormat(recordFormat);

                    // Aprire il file per l'aggiornamento
        myFile.open(AS400File.READ_WRITE, 0, AS400File.COMMIT_LOCK_LEVEL_NONE);

                    // Aggiornare il primo record del file.  Supponiamo
                    // che newName sia una Stringa con un nuovo nome per
                    // CUSTNAME
        Record updateRec = myFile.readFirst();
        updateRec.setField("CUSTNAME", newName);
        myFile.update(updateRec);

                    ....

                              // Chiudere il file in quanto è terminato l'utilizzo
```

```
myFile.close();

                          // Scollegarsi in quanto è terminato l'uso dell'accesso a livello record
sys.disconnectService(AS400.RECORDACCESS);
```

Utilizzare il metodowrite() per accodare i record alla fine di un file.Possono essere accodati ad un file sia una schiera di record che un record singolo.

Utilizzare il metodo deleteCurrentRecord() per cancellare il record che si trova nella posizione corrente del cursore.

Vincolo dei file:

Il programma Java può vincolare un file per impedire l'accesso al file da parte di altri utenti mentre il primo programma Java sta utilizzando il file.Seguono vari tipi di vincolo:

- Vincolo Lettura/Esclusivo - il programma Java corrente legge i record e nessun altro programma può accedere al file.
- Vincolo Lettura/Consentire lettura condivisa - il programma Java corrente legge i record e altri programmi possono leggere i record dal file.
- Vincolo Lettura/Consentire scrittura condivisa - il programma Java legge i record e altri programmi possono modificare il file.
- Vincolo Scrittura/Esclusivo - il programma Java corrente modifica il file e nessun altro programma può accedere al file.
- Vincolo Scrittura/Consentire lettura condivisa - il programma Java corrente modifica il file e altri programmi possono leggere i record dal file.
- Vincolo Scrittura/Consentire scrittura condivisa - il programma Java corrente modifica il file ed anche altri programmi possono modificare il file.

Per annullare i vincoli ottenuti tramite il metodo lock(), il programma Java avvia il metodo releaseExplicitLocks().

Utilizzo del blocco del record:

La classe AS400File utilizza il blocco del record per migliorare le prestazioni:

- Se il file viene aperto per l'accesso di sola lettura, un blocco di record viene letto quando il programma Java legge un record. Il blocco migliora le prestazioni dal momento che le successive richieste di lettura possono essere gestite senza accedere al server. Esistono delle leggere differenze di prestazione tra la lettura di un record singolo e la lettura di più record. Le prestazioni migliorano significativamente se i record sono accessibili al di fuori del blocco dei record memorizzati nella cache sul client.

  Il numero di record da leggere in ogni blocco può essere impostato quando si apre il file. Ad esempio:

```
                          // Creare un oggetto AS400, il file esiste su questo
                          // server.
AS400 sys = new AS400("mySystem.myCompany.com");

                          // Creare un oggetto file che rappresenta il file
SequentialFile myFile = new SequentialFile(sys, "/QSYS.LIB/MYLIB.LIB/MYFILE.FILE/%FILE%.MBR");

                          // Presupporre che la classe AS400FileRecordDescription sia
                          // stata usata per generare il codice di una sottoclasse di
                          // RecordFormat che rappresenta il formato record
                // del file MYFILE nella libreria MYLIB.  Il codice è stato
                                    // compilato ed è disponibile per essere usato dal programma
                                    // Java.
RecordFormat recordFormat = new MYFILEFormat();

                // Impostare il formato record per myFile.  E' necessario
                // farlo prima di richiamare open()
```

```
          myFile.setRecordFormat(recordFormat);

                                    // Aprire il file.
Specificare un fattore blocco di 50.
          int blockingFactor = 50;
          myFile.open(AS400File.READ_ONLY, blockingFactor, AS400File.COMMIT_LOCK_LEVEL_NONE);

                                   // Leggere il primo record del file.
Poiché
                                              // è stato specificato un fattore blocco, vengono reperiti 50
                                              // record durante questo richiamo di read().
          Record record = myFile.readFirst();
          for (int i = 1; i < 50 && record != null; i++)
     {

              // I record letti in questi loop opereranno fuori dal blocco di
              // record memorizzati nella cache sul client.
              record = myFile.readNext();
     }
                    ....

                                    // Chiudere il file in quanto è terminato l'utilizzo
          myFile.close();

                                              // Scollegarsi poiché è terminato l'utilizzo
                                              // dell'accesso a livello record
          sys.disconnectService(AS400.RECORDACCESS);
```

- Se il file viene aperto per l'accesso di sola scrittura, il fattore di blocco indica quanti record vengono scritti contemporaneamente sul file quando viene richiamato il metodo write(Record[]).

  Ad esempio:

```
                                    // Creare un oggetto AS400, il file esiste su questo
                                    // server.
     AS400 sys = new AS400("mySystem.myCompany.com");

                                    // Creare un oggetto file che rappresenta il file
          SequentialFile myFile = new SequentialFile(sys, "/QSYS.LIB/MYLIB.LIB/MYFILE.FILE/%FILE%.MBR");

                                    // Presupporre che la classe AS400FileRecordDescription sia
                                    // stata usata per generare il codice di una sottoclasse di
                                    // RecordFormat che rappresenta il formato record
                         // del file MYFILE nella libreria MYLIB.  Il codice è stato
                                              // compilato ed è disponibile per essere usato dal programma
                                              // Java.
          RecordFormat recordFormat = new MYFILEFormat();

                         // Impostare il formato record per myFile.  E' necessario
                                    // farlo prima di richiamare open()
          myFile.setRecordFormat(recordFormat);

                                    // Aprire il file.
Specificare un fattore blocco di 50.
          int blockingFactor = 50;
          myFile.open(AS400File.WRITE_ONLY, blockingFactor, AS400File.COMMIT_LOCK_LEVEL_NONE);

                                              // Creare una schiera di record da scrivere nel file
          Record[] records = new Record[100];
          for (int i = 0; i < 100; i++)
     {
                                              // Presupporre che il file disponga di due campi,
                                              // CUSTNAME e CUSTNUM
              records[i] = recordFormat.getNewRecord();
              records[i].setField("CUSTNAME", "Customer " + String.valueOf(i));
              records[i].setField("CUSTNUM", new Integer(i));
     }

                         // Scrivere i record nel file.  Poiché il
```

```
                                        // blocco è 50, vengono effettuati solo due trip
                                        // al server con singoli trip da 50 record
        myFile.write(records);

                  ....

                                    // Chiudere il file in quanto è terminato l'utilizzo
        myFile.close();

                                        // Scollegarsi poiché è terminato l'utilizzo
                                        // dell'accesso a livello record
        sys.disconnectService(AS400.RECORDACCESS);
```

- Se il file viene aperto come per l'accesso di sola scrittura, non viene creato alcun blocco. Ogni fattore di blocco specificato su open() viene ignorato.

Impostazione della posizione del cursore:

Un file aperto dispone di un cursore. Il cursore punta sul record da leggere, aggiornare o cancellare. Quando un file viene aperto per la prima volta il cursore punta sull'inizio del file. L'inizio del file si trova prima del primo record. Utilizzare i metodi che seguono per impostare la posizione del cursore:

- positionCursorAfterLast() - imposta il cursore dopo l'ultimo record.Questo metodo consente ai programmi Java di utilizzare il metodo readPrevious() per accedere ai record nel file.
- positionCursorBeforeFirst() - imposta il cursore prima del primo record.Questo metodo consente ai programmi Java di utilizzare il metodo readNext() per accedere ai record nel file.
- positionCursorToFirst() - imposta il cursore sul primo record.
- positionCursorToLast() - imposta il cursore sull'ultimo record.
- positionCursorToNext() - sposta il cursore sul record successivo.
- positionCursorToPrevious() - sposta il cursore sul record precedente.

L'esempio che segue mostra come utilizzare il metodo positionCursorToFirst() per posizionare il cursore.

```
                                    // Creare un oggetto AS400, il file esiste su questo
                                    // server.
    AS400 sys = new AS400("mySystem.myCompany.com");

                                    // Creare un oggetto file che rappresenta il file
        SequentialFile myFile = new SequentialFile(sys, "/QSYS.LIB/MYLIB.LIB/MYFILE.FILE/%FILE%.MBR");

                                    // Presupporre che la classe AS400FileRecordDescription sia
                                    // stata usata per generare il codice di una sottoclasse di
                                    // RecordFormat che rappresenta il formato record
                    // del file MYFILE nella libreria MYLIB.  Il codice è stato
                                        // compilato ed è disponibile per essere usato dal programma
                                        // Java.
        RecordFormat recordFormat = new MYFILEFormat();

                    // Impostare il formato record per myFile.  E' necessario
                    // farlo prima di richiamare open()
        myFile.setRecordFormat(recordFormat);

                                    // Aprire il file.
        myFile.open(AS400File.READ_WRITE, 1, AS400File.COMMIT_LOCK_LEVEL_NONE);

                                        // Si desidera cancellare il primo record del file.
        myFile.positionCursorToFirst();
        myFile.deleteCurrentRecord();

                  ....

                                    // Chiudere il file in quanto è terminato l'utilizzo
        myFile.close();
```

```
                                        // Scollegarsi poiché è terminato l'utilizzo
                                        // dell'accesso a livello record
        sys.disconnectService(AS400.RECORDACCESS);
```

Controllo di sincronizzazione:

Attraverso il controllo di sincronizzazione, il programma Java dispone di un altro livello di controllo sulla modifica di un file. Con il controllo di sincronizzazione attivo, le transazioni relative ad un file rimangono in sospeso fino a quando non vengono sottoposte a commit o rollback. Se si effettua il commit, tutte le modifiche vengono immesse nel file. Se si effettua il rollback, tutte le modifiche vengono eliminate. La transazione può modificare un record già esistente, aggiungere o cancellare un record o anche leggere un record a seconda del livello di blocco del controllo di sincronizzazione specificato in open().

I livelli del controllo di sincronizzazione sono i seguenti:

- All - ogni record ha cui si è avuto accesso nel file è vincolato fino a quando la transazione non viene sottoposta a commit o rollback.
- Change - i record aggiornati, aggiunti o cancellati nel file sono vincolati fino a quando la transazione non viene sottoposta a commit o rollback.
- Cursor Stability - i record aggiornati, aggiunti o cancellati nel file vengono vincolati fino a quando la transazione non viene sottoposta a commit o rollback.I record ai quali si è avuto accesso e che non sono stati modificati vengono vincolati fino a quando non si accede ad un altro record.
- None - non esiste controllo di sincronizzazione sul file.Le modifiche vengono immesse immediatamente nel file e non possono essere sottoposte a rollback.

E' possibile utilizzare il metodo startCommitmentControl() per avviare il controllo di sincronizzazione. Il controllo di sincronizzazione si applica al **collegamento** AS400. Una volta che il controllo di sincronizzazione viene avviato per un collegamento, esso si applica a tutti i file aperti in tale collegamento da quando il controllo è stato avviato. Sui file aperti prima dell'avvio del controllo di sincronizzazione non verrà applicato tale controllo. Il livello del controllo di sincronizzazione per file singoli viene specificato nel metodo open(). Specificare COMMIT_LOCK_LEVEL_DEFAULT per utilizzare lo stesso livello di controllo sincronizzazione specificato nel metodo startCommitmentControl().

Ad esempio:

```
                    // Creare un oggetto AS400, i file esistono su questo
                                            // server.
AS400 sys = new AS400("mySystem.myCompany.com");

                    // Creare tre oggetti file
SequentialFile myFile = new SequentialFile(sys, "/QSYS.LIB/MYLIB.LIB/MYFILE.FILE/%FILE%.MBR");
SequentialFile yourFile = new SequentialFile(sys, "/QSYS.LIB/YOURLIB.LIB/YOURFILE.FILE/%FILE%.MBR");
SequentialFile ourFile = new SequentialFile(sys, "/QSYS.LIB/OURLIB.LIB/OURFILE.FILE/%FILE%.MBR");

                    // Aprire yourFile prima di iniziare il controllo di sincronizzazione
                    // Non viene applicato alcun controllo di sincronizzazione a questo file. Le
                    // Il parametro del livello di blocco sincronizzazione viene ignorato
                    // perché il controllo di sincronizzazione non è avviato per il collegamento.
yourFile.setRecordFormat(new YOURFILEFormat());
yourFile.open(AS400File.READ_WRITE, 0, AS400File.COMMIT_LOCK_LEVEL_DEFAULT);

                    // Avviare il controllo sincronizzazione per il collegamento.
                    // Nota: uno qualsiasi di questi tre file potrebbe essere utilizzato per
                    // questa chiamata a startCommitmentControl().
    myFile.startCommitmentControl(AS400File.COMMIT_LOCK_LEVEL_CHANGE);

                    // Aprire myFile e ourFile
    myFile.setRecordFormat(new MYFILEFormat());
```

```
            // Utilizzare lo stesso livello di blocco sincronizzazione come
            // specificato all'avvio del controllo di sincronizzazione
myFile.open(AS400File.WRITE_ONLY, 0, COMMIT_LOCK_LEVEL_DEFAULT);

ourFile.setRecordFormat(new OURFILEFormat());
            // Specificare un livello di blocco sincronizzazione diverso da quello
            // specificato all'avvio del controllo di sincronizzazione
ourFile.open(AS400File.READ_WRITE, 0, COMMIT_LOCK_LEVEL_CURSOR_STABILITY);

            // scrivere e aggiornare i record in tutti e tre i file
          ....

            // Convalidare le modifiche per i file myFile e ourFile.
            // Notare che la sincronizzazione sincronizza tutte le modifiche del collegamento
            // anche se è stato richiamato solo su un oggetto AS400File.
myFile.commit();
            // Chiudere i file
myFile.close();
yourFile.close();
ourFile.close();

            // Chiudere il controllo sincronizzazione
            // Tale operazione chiude il controllo sincronizzazione per il collegamento.
ourFile.endCommitmentControl();

            // Scollegarsi in quanto è terminato l'uso dell'accesso a livello record
sys.disconnectService(AS400.RECORDACCESS);
```

Il metodo commit() sottopone a commit tutte le transazioni dall'ultimo limite di commit per il **collegamento**.Il metodo rollback() elimina tutte le transazioni dall'ultimo limite di commit per il **collegamento**.Il controllo di sincronizzazione per un collegamento viene terminato attraverso il metodo endCommitmentControl(). Se un file viene chiuso prima di chiamare il metodo commit() o rollback(), tutte le transazioni non sottoposte a commit vengono sottoposte a rollback. Tutti i file aperti durante il controllo di sincronizzazione devono essere chiusi prima che sia chiamato il metodo endCommitmentControl().

L'esempio che segue mostra come avviare il controllo di sincronizzazione, la funzione di commit o rollback, quindi terminare il controllo di sincronizzazione:

```
          // ... presupporre che l'oggetto AS400 e il file
                                  // siano stati avviati.

            // Avviare il controllo sincronizzazione per *CHANGE
aFile.startCommitmentControl(AS400File.COMMIT_LOCK_LEVEL_CHANGE);

            // ... aprire il file ed effettuare alcune modifiche. Ad
            // esempio, aggiornare, aggiungere o cancellare i record.

            // In base ad un indicatore, salvare o eliminare le
            // transazioni.
if (saveChanges)
      aFile.commit();
                  else
      aFile.rollback();

            // Chiudere il file
aFile.close();

            // Chiudere il controllo sincronizzazione per il collegamento.
aFile.endCommitmentControl();
```

ServiceProgramCall

La classe ServiceProgramCall consente di richiamare un programma di servizio iSeries.ServiceProgramCall è una sottoclasse della classe ProgramCall che si utilizza per richiamare i programmi iSeries.Se si desidera richiamare un programma iSeries, utilizzare la classe ProgramCall.

La classe ServiceProgramCall consente di richiamare un programma di servizio iSeries, spostare dati su un programma di servizio iSeries attraverso parametri di immissione e accedere ai dati che il programma di servizio iSeries restituisce attraverso parametri di emissione. L'utilizzo di ServiceProgramCall fa si che l'oggetto AS400 si colleghi a iSeries. Consultare la gestione dei collegamenti per informazioni sulla gestione dei collegamenti.

La funzionalità predefinita per i programmi di servizio è quella di essere eseguiti su un lavoro server separato, anche quando il programma Java e il programma di servizio si trovano sullo stesso server. E' possibile sostituire la funzionalità predefinita e fare in modo che i programma di servizio siano eseguiti nel lavoro Java utilizzando il metodo ereditato (da ProgramCall) setThreadSafe().

Utilizzo della classe ServiceProgramCall

Per utilizzare la classe ServiceProgramCall, è necessario assicurarsi di soddisfare i requisisti che seguono:

- Il programma di servizio deve trovarsi su un iSeries
- E' possibile passare non più di sette parametri al programma di servizio
- Il valore di ritorno del programma di servizio è void o un valore numerico

Gestione degli oggetti ProgramParameter

La classe ProgramParameter gestisce la classe ServiceProgramCall per passare dati parametro a e da un programma di servizio iSeries.E' possibile passare dati di immissione ad un programma iSeries service con setInputData().

E' possibile richiedere la quantità di dati di emissione che si desiderano avere restituiti con setOutputDataLength().E' possibile richiamare i dati di emissione dopo che il programma di servizio ha completato l'esecuzione con getOutputData().Oltre ai dati in se stessi, ServiceProgramCall ha bisogno di sapere come passare i dati parametro al programma di servizio. Il metodo setParameterType() di ProgramParameter viene utilizzato per fornire queste informazioni. Il tipo indica se il parametro viene passato per valore o per riferimento. In entrambi i casi, i dati vengono inviati dal client al server. Una volta che i dati si trovano su iSeries, il server utilizza il tipo di parametri per richiamare correttamente il programma di servizio.

Tutti i parametri saranno nel modulo di una schiera di byte. Perciò, per convertire tra i formati iSeries e Java, è necessario utilizzare le classi conversione dati e descrizione.

Classi SystemStatus

Le classi SystemStatus consentono di richiamare le informazioni dello stato di sistema e richiamare e modificare le informazioni del lotto del sistema.L'oggetto SystemStatus consente di richiamare le informazioni dello stato di sistema incluso quanto segue:

- getUsersCurrentSignedOn(): restituisce il numero di utenti collegati attualmente sul sistema
- getUsersTemporarilySignedOff(): restituisce il numero di lavori interattivi che sono scollegati
- getDateAndTimeStatusGathered(): restituisce la data e l'orario in cui le informazioni sullo stato del sistema sono state raccolte
- getJobsInSystem(): restituisce il numero totale di lavori utente e sistema che sono attualmente in esecuzione

- getBatchJobsRunning(): restituisce il numero di lavori batch che sono attualmente in esecuzione sul sistema
- getBatchJobsEnding(): restituisce il numero di lavori batch che si trovano nel processo di conclusione
- getSystemPools(): restituisce una enumerazione che contiene un oggetto SystemPool per ogni lotto del sistema

In aggiunta ai metodi della classe SystemStatus class, è inoltre possibile accedere a SystemPool attraverso SystemStatus.SystemPool consente di richiamare informazioni relative ai lotti del sistema e modificare informazioni del lotto del sistema.

Esempio

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

L'esempio che segue mostra come utilizzare la memorizzazione nella cache con la classe SystemStatus:

```
AS400 system = new AS400("MyAS400");
SystemStatus status = new SystemStatus(system);

// Attivare la cache. Essa è disattivata per impostazione predefinita.
status.setCaching(true);

// Ciò reperirà il valore dal sistema.
// Ogni chiamata successiva userà il valore memorizzato nella cache
// invece di reperirlo dal sistema.
int jobs = status.getJobsInSystem();


// ... Eseguire altre operazioni in questo punto ...


// Ciò determina se la cache è ancora abilitata.
if (status.isCaching())
{
    // Ciò reperirà il valore dalla cache.
  jobs = status.getJobsInSystem();
}

// Successivamente andare al sistema, indipendentemente dal fatto che la cache sia abilitata o meno.
status.refreshCache();

// Ciò reperirà il valore dal sistema.
jobs = status.getJobsInSystem();

// Disattivare la cache. Ogni chiamata successiva andrà al sistema.
status.setCaching(false);

// Ciò reperirà il valore dal sistema.
jobs = status.getJobsInSystem();
```

Classe SystemPool:

La classe SystemPool consente di richiamare e modificare informazioni del lotto del sistema incluso quanto segue:

- Il metodo getPoolSize() restituisce la dimensione del lotto e il metodosetPoolSize() imposta la dimensione del lotto.
- Il metodo getPoolName() richiama il nome del lotto e il metodo setPoolName() imposta il nome del lotto.
- Il metodo getReservedSize() restituisce la quantità di memoria nel lotto riservata per l'utilizzo del sistema.
- Il metodo getDescription() restituisce la descrizione del lotto del sistema.

- Il metodo getMaximumActiveThreads() restituisce il numero massimo di sottoprocessi che possono essere attivi nel lotto in qualsiasi momento.
- Il metodo setMaximumFaults() imposta le guide massime di errori al secondo da utilizzare nel lotto del sistema.
- Il metodo setPriority() imposta la priorità di questo lotto del sistema relative alle priorità di altri lotti del sistema.

Esempio

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
//Creare l'oggetto AS400.
AS400 as400 = new AS400("system name");

//Creare un oggetto lotto di sistema.
SystemPool systemPool = new SystemPool(as400,"*SPOOL");

//Richiamare l'opzione di paginazione del lotto di sistema
System.out.println("Paging option : "+systemPool.getPagingOption());
```

SystemValues

Le classi SystemValues consentono al programma Java di richiamare e modificare i valori di sistema e gli attributi di rete. E' inoltre possibile definire il gruppo che contiene i valori di sistema che si desiderano.

Un oggetto SystemValue contiene anzitutto le informazioni che seguono:
- Nome
- Descrizione
- Release
- Valore

Utilizzando la classe SystemValue, richiamare un valore di sistema singolo utilizzando il metodo getValue() e modificare il valore di sistema utilizzando il metodosetValue().

E' inoltre possibile richiamare gruppi di informazioni relative ad un valore di sistema particolare:
- Per richiamare il gruppo definito dal sistema a cui appartiene il valore di sistema, utilizzare il metodo getGroup().
- Per richiamare il gruppo definito dall'utente a cui appartiene l'oggetto SystemValue (se esiste), utilizzare i metodi getGroupName() e getGroupDescription().

Quando il valore di un valore di sistema viene richiamato per la prima volta, il valore viene richiamato da iSeries e memorizzato nella cache. Nei recuperi successivi, viene restituito il valore memorizzato nella cache. Se il valore iSeries corrente è quello che si desidera invece del valore memorizzato in cache, deve essere eseguito clear() per eliminare il contenuto corrente della cache.

SystemValueList

SystemValueList rappresenta un elenco di valori di sistema sul server iSeries specificato. L'elenco è diviso in diversi gruppi definiti dal sistema che consentono al programma Java di accedere ad una parte dei valori di sistema alla volta.

Gruppo valori di sistema

SystemValueGroup rappresenta una raccolta definita dall'utente di valori di sistema e attributi di rete.Più che un contenitore, è un sistema di produzione per la creazione e il mantenimento di raccolte univoche di valori di sistema.

E' possibile creare SystemValueGroup specificando uno dei gruppi definiti dal sistema (una delle costanti nella classe SystemValueList) o specificando una schiera di nomi valore di sistema.

E' possibile aggiungere individualmente i nomi dei valori di sistema da includere nel gruppo utilizzando il metodo add().E' inoltre possibile rimuoverli utilizzando il metodo remove().

Una volta popolato SystemValueGroup con i nomi dei valori di sistema richiesti, richiamare gli oggetti SystemValue effettivi dal gruppo chiamando il metodo getSystemValues(). In questo modo un oggetto SystemValueGroup prende una serie di nomi valore di sistema e crea un vettore degli oggetti SystemValue, tutti comprensivi della descrizione del sistema, del nome gruppo e del gruppo di SystemValueGroup.

Per aggiornare in una volta un vettore negli oggetti SystemValue, utilizzare il metodo refresh().

Esempi di utilizzo delle classi SystemValue e SystemValueList

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

L'esempio che segue mostra come creare e richiamare un valore di sistema:

```
//Creare un oggetto AS400
AS400 sys = new AS400("mySystem.myCompany.com");

//Creare un valore di sistema che rappresenti il secondo corrente sul sistema.
SystemValue sysval = new SystemValue(sys, "QSECOND");

//Reperire il valore.
String second = (String)sysval.getValue();

//A questo punto QSECOND è memorizzato nella cache. Ripulire la cache per reperire il
//valore più aggiornato dal sistema.
sysval.clear();
second = (String)sysval.getValue();

//Creare un elenco di valori di sistema.
SystemValueList list = new SystemValueList(sys);

//Reperire tutti i valori di sistema data/ora.
Vector vec = list.getGroup(SystemValueList.GROUP_DATTIM);

//Scollegarsi dal sistema.
sys.disconnectAllServices();
```

esempi di utilizzo della classe SystemValueGroup

L'esempio che segue mostra come creare un gruppo di nomi valore di sistema e quindi gestirli:

```
//Creare un oggetto AS400
AS400 sys = new AS400("mySystem.myCompany.com");

//Creare un gruppo di valori di sistema che rappresenti inizialmente tutti gli attributi di rete sul sistema.
String name = "My Group";
String description = "This is one of my system values.";
SystemValueGroup svGroup = new SystemValueGroup(sys, name, description, SystemValueList.GROUP_NET);

//Aggiungere altri nomi di valori di sistema al gruppo e eliminare quelli non desiderati.
svGroup.add("QDATE");
svGroup.add("QTIME");
svGroup.remove("NETSERVER");
svGroup.remove("SYSNAME");

//Ottenere gli oggetti SystemValue reali. Essi vengono restituiti in un vettore.
Vector sysvals = svGroup.getSystemValues();
```

```
//Si noterà che questo è uno dei valori di sistema personali.
SystemValue mySystemValue = (SystemValue)sysvals.elementAt(0);
System.out.println(mySystemValue.getName()+" - "+mySystemValue.getGroupDescription());

//E' possibile aggiungere un altro oggetto SystemValue da un altro sistema nel gruppo.
AS400 sys2 = new AS400("otherSystem.myCompany.com");
SystemValue sv = new SystemValue(sys2, "QDATE");
sysvals.addElement(sv);

//Ora aggiornare contemporaneamente l'intero gruppo di valori di sistema.
//Non importa se alcuni valori di sistema provengono da server iSeries differenti.
//Non importa se alcuni valori di sistema sono stati generati usando SystemValueGroup e altri no.
SystemValueGroup.refresh(sysvals);

//Scollegarsi dai sistemi.
sys.disconnectAllServices();
sys2.disconnectAllServices();
```

Classe Trace

La classe Trace consente al programma Java di registrare i punti di traccia ed i messaggi di diagnostica. Queste informazioni aiutano a riprodurre e diagnosticare i problemi

Nota: è possibile inoltre impostare la traccia utilizzando le proprietà di sistema traccia.

La classe Trace registra le categorie di informazioni che seguono:

Categoria di informazioni	Descrizione
Conversione	Registra le conversioni della serie di caratteri tra le pagine Unicode e di codice. Questa categoria viene utilizzata solo dalle classi IBM Toolbox per Java.
Flusso di dati	Registra i dati che si trovano nei programmi iSeries e Java. Questa categoria viene utilizzata solo dalle classi IBM Toolbox per Java.
Diagnostica	Registra le informazioni sullo stato
Errore	Registra errori aggiuntivi che causano un'eccezione
Informazioni	Traccia il flusso attraverso un programma.
PCML	Questa categoria viene utilizzata per stabilire come PCML interpreta i dati che vengono inviati al e dal server.
Proxy	Questa categoria viene utilizzata dalle classi IBM Toolbox per Java per registrare il flusso di dati tra il client e il server proxy.
Avvertenza	Registra informazioni relative agli errori che il programma è stato capace di recuperare.
All	Questa categoria viene utilizzare per abilitare o disabilitare l'esecuzione della traccia di tutte le categorie in una volta. Non è possibile registrare le informazioni di traccia direttamente in questa categoria.

Le classi IBM Toolbox per Java utilizzano inoltre le categorie di traccia. Quando un programma Java abilita la registrazione, le informazioni IBM Toolbox per Java vengono incluse con le informazioni registrate dall'applicazione.

E' possibile abilitare la traccia per una categoria singola o una serie di categorie.Una volta selezionate le categorie, utilizzare il metodo setTraceOn per attivare o disattivare la traccia.I dati vengono scritti sulla registrazione utilizzando il metodo log.

E' possibile inviare dati traccia per diversi componenti in registrazioni separate. I dati traccia, per impostazione predefinita, vengono scritti nella registrazione predefinita. Utilizzare la traccia del componente per scrivere dati traccia specifici all'applicazione in una registrazione separata o un'emissione standard. Utilizzando la traccia del componete, è possibile separare facilmente dati traccia per un'applicazione specifica da altri dati.

Una registrazione eccessiva potrebbe influire sulle prestazioni. Utilizzare il metodo isTraceOn per interrogare lo stato corrente della traccia. Il programma Java può utilizzare questo metodo per stabilire se crea il record di traccia prima di chiamare il metodo di registrazione. Richiamare il metodo di registrazione mentre la registrazione non è attiva, non è un errore ma ci vuole più tempo.

Il valore predefinito è quello di scrivere informazioni di registrazione standard. Per reindirizzare la registrazione su un file, richiamare il metodosetFileName() dall'applicazione Java. In generale ciò funziona solo per le applicazioni Java perché la maggior parte dei browser non forniscono accesso applet per scrivere sul file system locale.

La registrazione viene disattivata per impostazione predefinita. Programmi Java forniscono all'utente un modo di attivare la registrazione che renda semplice l'operazione. Ad esempio, l'applicazione può analizzare un parametro della riga comandi che indica quale categoria di dati viene registrata. L'utente può impostare questo parametro quando sono necessarie informazioni di registrazione.

Esempi

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

Gli esempi che seguono mostrano come utilizzare la classe Trace

Esempio di utilizzo di setTraceOn() e di scrittura dei dati in una registrazione utilizzando il metodo della registrazione

```
// Abilitare la registrazione diagnostica, delle informazioni e di avvertenza.
Trace.setTraceDiagnosticOn(true);
Trace.setTraceInformationOn(true);
Trace.setTraceWarningOn(true);

// Attivare la traccia.
Trace.setTraceOn(true);

// ... A questo punto nel programma Java, scrivere nella registrazione.
Trace.log(Trace.INFORMATION, "Just entered class xxx, method xxx");

// Disattivare la traccia.
Trace.setTraceOn(false);
```

Esempio: utilizzo di Trace

Nel seguente codice, il Metodo 2 è il modo più opportuno di utilizzare Trace.

```
// Metodo 1 - creare un record di traccia
// quindi chiamare il metodo di registrazione e fare in modo che la classe trace
// determini se i dati devono essere registrati. Questa operazione funzionerà ma sarà
// più lenta del seguente codice.
String traceData = new String("Just entered class xxx, data = ");
   traceData = traceData + data + "state = " + state;
   Trace.log(Trace.INFORMATION, traceData);

// Metodo 2 - controllare lo stato registrazione prima di inserire le informazioni nella
```

```
     // registrazione. Ciò si rivela più veloce quando la traccia non è attiva.
if (Trace.isTraceOn() && Trace.isTraceInformationOn())
{
           String traceData = new String("just entered class xxx, data = ");
           traceData = traceData + data + "state = " + state;
           Trace.log(Trace.INFORMATION, traceData);
}
```

Esempio: utilizzo della traccia del componente

```
// Creare un stringa componente. E' più efficace creare un
     // oggetto rispetto a molte costanti letterali String.
     String myComponent1 = "com.myCompany.xyzComponent";
     String myComponent2 = "com.myCompany.abcComponent";

// Inviare i dati di IBM Toolbox per Java e della traccia del componente ognuno a file separati.
// La traccia conterrà tutte le informazioni di traccia, mentre ogni
     // file registrazione componente conterrà informazioni di traccia specifiche per
     // tale componente. Se non viene specificato un file traccia, tutti i dati traccia
// verranno inseriti nell'emissione standard con il componente specificato davanti
     // ad ogni messaggio di traccia.

     // Trace.setFileName("c:\\bit.bucket");
     // Trace.setFileName(myComponent1, "c:\\Component1.log");
     // Trace.setFileName(myComponent2, "c:\\Component2.log");

     Trace.setTraceOn(true);             // Attivare la traccia.
     Trace.setTraceInformationOn(true);  // Abilitare i messaggi informativi.

// Registrare i dati di traccia specifici per il componente o dati di traccia di IBM Toolbox
     // per Java generali.

     Trace.setFileName("c:\\bit.bucket");
     Trace.setFileName(myComponent1, "c:\\Component1.log");
```

Utenti e gruppi

Le classi user e group consentono di richiamare un elenco di utenti e gruppi di utenti sul server iSeries ed anche informazioni su ogni utente tramite un programma Java.

Nota: IBM Toolbox per Java fornisce inoltre classi resource che presentano un framework generica ed un'interfaccia di programmazione coerente per gestire vari oggetti ed elenchi iSeries. Dopo aver letto informazioni relative alle classi nel pacchetto access e nel pacchetto resource, è possibile scegliere l'oggetto più adatto all'applicazione.Le classi Resource per gestire gli utenti sono RUser e RUserList.

Alcune informazioni utente che è possibile richiamare includono la data dell'ultimo collegamento, lo stato, la data in cui è stata modificata la parola d'ordine, la data di scadenza della parola d'ordine e la classe utente. Quando si accede all'oggetto User, utilizzare il metodo setSystem() per impostare il nome di sistema ed il metodo setName() per impostare il nome utente. Dopo queste operazioni, si utilizza il metodo loadUserInformation() per richiamare le informazioni da iSeries.

L'oggetto UserGroup rappresenta un utente speciale il cui profilo utente è un profilo di gruppo.Utilizzando il metodo getMembers(), è possibile venga restituito un elenco di utenti membri del gruppo.

Il programma Java può scorrere l'elenco tramite una enumerazione. Tutti gli elementi nella enumerazione sono oggetti User ad esempio:

```
     // Creare un oggetto AS400.
   AS400 system = new AS400 ("mySystem.myCompany.com");

   // Creare l'oggetto UserList.
          UserList userList = new UserList (system);
```

```
    // Richiamare l'elenco di tutti gli utenti e gruppi.
            Enumeration enum = userList.getUsers ();

    // Iterare l'elenco.
            while (enum.hasMoreElements ())
{
                    User u = (User) enum.nextElement ();
            System.out.println  (u);
}
```

Richiamo di informazioni su utenti e gruppi

Utilizzare uno UserList per richiamare un elenco di quanto segue:

- Tutti gli utenti e gruppi
- Solo gruppi
- Tutti gli utenti membri dei gruppi
- Tutti gli utenti non membri di gruppi

La sola proprietà dell'oggetto UserList che è necessario impostare corrisponde all'oggetto AS400 che rappresenta il sistema da cui l'elenco di utenti deve essere richiamato.

Per impostazione predefinita, vengono restituiti tutti gli utenti. Utilizzare una combinazione di setUserInfo() e setGroupInfo() per specificare esattamente quali utenti vengono vengono restituiti.

Esempio: utilizzo di UserList per elencare tutti gli utenti in un dato gruppo.

Classe UserSpace

La classe UserSpace rappresenta un'area utente sul server.I parametri richiesti sono il nome dell'area utente e l'oggetto AS400 che rappresenta il server su cui si trova l'area utente.Esistono metodi nella classe UserSpace per eseguire le operazioni riportate di seguito:

- Creare un'area utente.
- Cancellare un'area utente.
- Leggere da un'area utente.
- Scrivere in un'area utente.
- Richiamare gli attributi di uno spazio utente. Un programma Java può richiamare il valore iniziale, il valore lunghezza e gli attributi estensibili automaticamente di uno spazio utente.
- Impostate gli attributi di uno spazio utente. Un programma Java può impostare il valore iniziale, il valore lunghezza e gli attributiestensibili automaticamente di un'area utente.

L'oggetto UserSpace richiede il nome del percorso IFS (Integrated File System) del programma. Per ulteriori informazioni, consultare i nomi percorso IFS (Integrated file system).

L'utilizzo della classe UserSpace collega l'oggetto AS400 al server. Consultare la gestione dei collegamenti per informazioni sulla gestione dei collegamenti.

Il seguente esempio crea uno spazio utente, quindi vi scrive dati.

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
                    // Creare un oggetto AS400.
AS400 sys = new AS400("mySystem.myCompany.com");

                    // Creare un oggetto spazio utente.
UserSpace US = new UserSpace(sys,
          "/QSYS.LIB/MYLIB.LIB/MYSPACE.USRSPC");
```

```
                    // Usare il metodo create per creare uno spazio utente
                    // sul server.

US.create(10240,                          // La dimensione iniziale è 10KB
          true,                           // Sostituire se lo spazio utente esiste già
          " ",                            // Nessun attributo esteso
          (byte) 0x00,                    // Il valore iniziale è null
          "Created by a Java program",    // La descrizione dello spazio utente
          "*USE");                        // Il pubblico ha l'autorizzazione all'uso allo spazio utente


                    // Usare il metodo write per scrivere i byte nello spazio utente.
     US.write("Write this string to the user space.", 0);
```

Classi Commtrace

Le classi commtrace di IBM Toolbox per Java consentono ai programmi Java di gestire dati relativi alla traccia della comunicazioni per una descrizione di linea LAN (Ethernet o token ring) specificata. Il pacchetto commtrace include una classe che è possibile eseguire come programma di utilità autonomo per formattare dati di traccia delle comunicazioni.

Quando si esegue il dump di una traccia delle comunicazioni per un server iSeries in un file di flusso, le informazioni sono salvate in formato binario. Le classi commtrace consentono all'utente di gestire i vari componenti del file di flusso.

Nota: i file di traccia delle comunicazioni possono contenere informazioni riservate, ad esempio, parole d'ordine non codificate. Quando un file di traccia delle comunicazioni si trova nel server iSeries, solo utenti con autorizzazione speciale *SERVICE possono accedere ai dati di traccia. Se si sposta il file in un client, accertarsi di proteggere il file in maniera appropriata. Per ulteriori informazioni sulle tracce delle comunicazioni, consultare i collegamenti alla fine di questa pagina.

utilizzare le classi commtrace per svolgere le seguenti attività:

- Formattare i dati di traccia grezzi.
- Analizzare qualsiasi dato per estrarre le informazioni desiderate. E' possibile analizzare sia i dati grezzi che i dati formattati, ammesso che si siano utilizzate le classi commtrace per formattare i dati.

Per una rappresentazione visiva che mostri in che modo le commtrace rappresentano le strutture in un file di traccia delle comunicazioni, consultare la pagina seguente:

"Modello Commtrace" a pagina 179

Il pacchetto commtrace include le seguenti classi:

"Classi Format e FormatProperties" a pagina 181: La classe Format legge sia i dati grezzi che i dati formattati da una traccia delle comunicazioni. FormatProperties imposta le proprietà per l'oggetto Format, come ad esempio le ore di avvio e di fine, gli indirizzi IP, le porte e così via.

Nota: è possibile anche eseguire la classe Format come programma autonomo.

"Classe Prolog" a pagina 184: richiama informazioni dalla sezione iniziale a 256 byte di una traccia delle comunicazioni del server iSeries.

"Classe Frame" a pagina 185: richiama informazioni sulle frame della traccia delle comunicazioni.

"Classe LanHeader" a pagina 185: richiama informazioni dalla sezione dei dati che si presenta una volta, accanto all'inizio di una frame. Questa sezione in genere contiene informazioni specifiche sull'hardware che includono informazioni generali sulla frame, come ad esempio il numero della frame, la lunghezza dei dati e così via.

"Classe IPPacket" a pagina 186: richiama informazioni dai dati nel pacchetto. IPPacket è una classe astratta principale per i tipi differenti di pacchetti di dati che sono supportati dal pacchetto commtrace.

"Classe Header" a pagina 187: richiama le informazioni dall'intestazione del pacchetto e dai dati associati. Header è la classe astratta principale per i tipi differenti di intestazioni di pacchetto che sono supportate dal pacchetto commtrace.

La maggior parte delle classi restanti nel pacchetto com.ibm.as400.commtrace sono specifiche per il tipo di dati di traccia che si desidera gestire. Per ulteriori informazioni sulle tracce delle comunicazioni e su tutte le classi commtrace, fare riferimento alle seguenti pagine:

Documentazione di riferimento Javadoc

Traccia delle comunicazioni

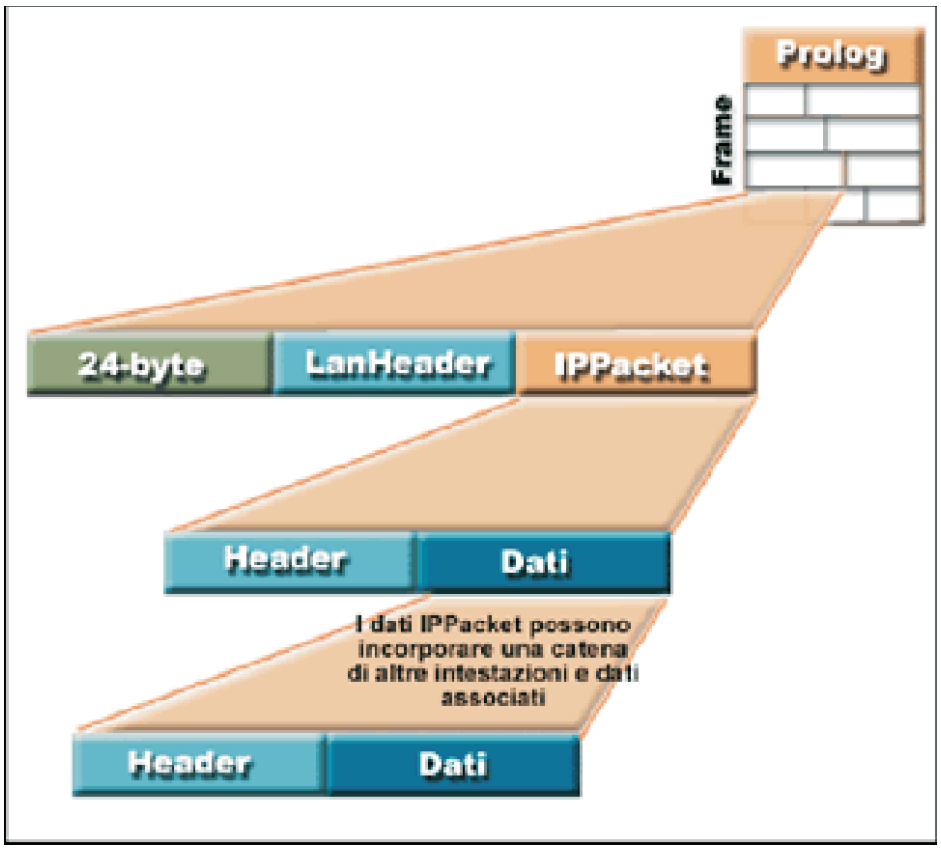
Modello Commtrace

La seguente illustrazione mostra come le classi commtrace corrispondano ad un file di traccia delle comunicazioni. Il grafico indica anche le convenzioni di denominazione utilizzate dalle classi commtrace per i componenti in una traccia delle comunicazioni.

Prolog: una sezione iniziale a 256 byte di una traccia delle comunicazioni per una descrizione di linea LAN. Il Prolog contiene informazioni generali sulla traccia, come ad esempio le ore di avvio e di fine, il numero di byte raccolti e così via.

Frame: Record di grandezza variabile che contengono i dati che il server iSeries ha trasmesso durante la traccia delle comunicazioni.

Figura 1: modello Commtrace



Ogni Frame nel file di traccia contiene due sezioni iniziali (che forniscono informazioni generali sul contenuto della frame) ed il pacchetto trasmesso dal server iSeries tra sé stesso ed un punto differente sulla rete.

La prima sezione in assoluto di dati a 24-byte contiene informazioni generali sul contenuto della frame, come ad esempio il numero frame e la lunghezza dei dati. Utilizzare la classe Frame per elaborare queste informazioni.

LanHeader: una sezione di dati che ricorre una sola volta in ogni frame (dopo i 24-byte iniziali di dati) e contiene informazioni generali sull'IPacket che segue.

IPPacket: una sezione di dati, composta da una o più intestazioni e dai dati associati. IPPacket rappresenta tutti i pacchetti di dati che la rete ha trasmesso per questa frame durante la traccia delle comunicazioni. IPPacket è una classe astratta quindi si utilizzeranno le varie sottoclassi concrete per elaborare le intestazioni ed i dati nei pacchetti.

Header: una sezione di dati all'inizio di un IPPacket che descrive i seguenti dati del pacchetto e, se opportuno, punta alla successiva intestazione. Nel pacchetto commtrace, Header è una classe astratta, quindi si utilizzeranno le varie sottoclassi concrete per elaborare i dati.

Descrizione lunga di Figura 1: modello Commtrace (rzahh587.gif):

In IBM Toolbox per Java: classi Commtrace

Questa figura illustra in termini generici come le classi commtrace corrispondano ad un file di traccia delle comunicazioni.

Descrizione

La figura è composta da quanto segue:

- Un'immagine di forma quadrata sullo sfondo di destra etichettata 'Prolog'. Questo quadrato è diviso in righe che rappresentano una sezione di una traccia di comunicazioni per la descrizione di una linea LAN. Scritta in verticale sul lato sinistro dell'immagine vi è la parola 'Frame'
- Un'immagine in primo piano sulla sinistra che rappresenta un rettangolo orizzontale. Questo rettangolo è collegato al quadrato principale sullo sfondo da linee su ciascun lato. Il rettangolo rappresenta una frame nel file di traccia ed è diviso in 3 sezioni:
  - Una sezione verde etichettata 24-byte che rappresenta la sezione di dati che contengono informazioni generali sul contenuto della frame.
  - Una sezione blu chiaro etichettata LanHeader che indica la sezione di dati che contiene informazioni generali sull'IPacket che segue.
  - Una sezione di colore rossiccio etichettata IPPacket che indica la sezione di dati composta di una o più intestazioni e dei dati associati.
- Un altro rettangolo orizzontale è posto sotto al primo. Una linea su ciascun lato del rettangolo lo collega all'IPPacket. Il rettangolo rappresenta una versione dettagliata della sezione IPPacket e consiste di 2 sezioni:
  - Una sezione blu chiaro etichettata Intestazione che indica la sezione di dati all'inizio di un IPPacket
  - Una sezione blu scuro etichettata Dati che indica i dati del pacchetto e punta alla successiva intestazione.
- Un terzo rettangolo orizzontale viene visualizzato sotto la sezione Dati blu scuro. Una linea su ciascun lato del rettangolo lo collega alla sezione Dati. Il rettangolo rappresenta le 2 sezioni dell'IPPacket e riporta 'Informazioni IPPacket incorporano una catena di altre intestazioni e dati associati':
-
  - Una sezione blu chiaro etichettata Intestazione che indica la sezione di dati all'inizio di un IPPacket
  - Una sezione blu scuro etichettata Dati che indica i dati del pacchetto e punta alla successiva intestazione.

Una frame (primo rettangolo orizzontale) in un file di traccia (immagine di sfondo) contiene 2 sezioni di dati, la sezione a 24-byte (verde) di dati e la LanHeader (blu chiaro), insieme al pacchetto (rossiccio).

L'IPPacket (secondo rettangolo orizzontale) elaborato nella frame consiste dell'Intestazione (blu chiaro) e dei Dati (blu scuro) che a loro volta puntano ad un'altra intestazione e dati (terzo rettangolo orizzontale). Il secondo ed il terzo rettangolo sono uniti da linee di collegamento su ciascun lato dalla sezione Dati e di nuovo all'IPPacket il che indica la catena continua per le intestazioni e i dati associati in una traccia di comunicazioni.

Classi Format e FormatProperties

La classe Format funge da interfaccia tra il programma di chiamata e le frame della traccia. La classe FormatProperties consente di impostare e richiamare proprietà che determinano come funziona l'oggetto Format quando riscontra informazioni nelle Frame della traccia.

Classe Format

Utilizzare la classe Format per leggere sia dati di traccia grezzi che dati di traccia già formattati utilizzando le classi commtrace.

Nota: non è possibile utilizzare le classi commtrace per leggere una traccia delle comunicazioni formattata utilizzando il comando CL (control language) PRTCMNTRC (Stampa traccia comunicazioni).

utilizzare la classe Format per analizzare e formattare le informazioni in una traccia, quindi inviare tali informazioni formattate in un file o in un'unità di stampa. Inoltre, si potrebbe voler creare un front end grafico che visualizzi le informazioni in un'applicazione autonoma o entro un browser. Quando si desidera selezionare solo dati specifici, utilizzare la classe Format per fornire tali informazioni al programma Java. Ad esempio, si potrebbe utilizzare la classe Format per leggere indirizzi IP estraendoli da una traccia ed utilizzare quindi tali dati nel programma.

I programmi di creazione di Format accettano argomenti che rappresentano dati non formattati, come ad esempio un oggetto IFSFileInputStream, un file locale o il file di traccia binario. Per visualizzare una traccia che è già stata formattata, utilizzare il programma di creazione di Format predefinito, quindi utilizzare Format.openIFSFile() o Format.openLclFile() per specificare il file formattato che si desidera utilizzare.

Esempi

I seguenti esempi mostrano come è possibile visualizzare una traccia salvata o formattare una traccia binaria.

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

Esempio: visualizzazione di una traccia salvata

```
Format fmt = new Format();
fmt.openLclFile("/path/to/file");

// Leggere il Prolog
System.out.println(fmt.getRecFromFile());
// Il numero totale di record nella traccia TCP e non TCP
System.out.println("Total Records:" + fmt.getIntFromFile());
String rec;
// Leggere i record fino alla fine.
while((rec = fmt.getRecFromFile())!=null) {
System.out.println(rec);
```

Esempio: formattazione di una traccia binaria

```
// Creare una FormatProperties. Per impostazione predefinita visualizza tutti i dati.
FormatProperties fmtprop = new FormatProperties();


Format fmt = new Format("/path/to/file");
// Imposta le proprietà del filtro per questo formato
fmt.setFilterProperties(fmtprop);
fmt.setOutFile("/path/to/output/file");
// Formattare il prolog
fmt.formatProlog();
// Formattare la traccia ed inviare i dati al file specificato
fmt.toLclBinFile();
```

Esecuzione di Format come un programma di utilità autonomo

E' possibile anche eseguire la classe Format come programma di utilità autonomo. Per ulteriori informazioni, consultare il seguente argomento:

Esecuzione di Format come un programma autonomo

Classe FormatProperties

Utilizzare la classe FormatProperties per specificare e richiamare le proprietà per l'oggetto Format. In altre parole, quando si utilizza la classe Format per inviare informazioni ad un file, utilizzare la classe FormatProperties per filtrare le informazioni che si desidera inviare.

Queste proprietà specificano come si desidera che l'oggetto Format gestisca le informazioni che riscontra nelle Frame della traccia delle comunicazioni. La funzionalità predefinita per l'oggetto Format consiste nell'ignorare le proprietà per cui non è stato fornito un valore specifico.

La classe FormatProperties fornisce delle costanti che si utilizzano per impostare proprietà. L'impostazione delle proprietà consente all'oggetto Format di verificare quali filtri l'utente desidera utilizzare. Ad esempio, il seguente codice imposta un oggetto Format per visualizzare una finestra di dialogo di avanzamento e non visualizzare le frame di trasmissione:

```
FormatProperties prop = new FormatProperties();
prop.setProgress(FormatProperties.TRUE);
prop.setBroadcast(FormatProperties.NO);
```

La maggior parte delle proprietà sono disponibili per l'oggetto Format come filtri che si impostano per includere esplicitamente dati specifici. Una volta impostati i filtri, l'oggetto Format visualizza solo i dati che corrispondono a tali filtri. Ad esempio, il seguente codice imposta un filtro per visualizzare le frame che si sono presentate tra tra un'ora di avvio e di fine particolare:

```
FormatProperties prop = new FormatProperties();
  // Impostare il filtro sulle ore di avvio e di fine del 22 Luglio, 2002,
  // 2:30 p.m. e 2:45 p.m. GMT.
  // L'ora viene espressa come una registrazione data/ora Unix(TM), che si
  // si basa sulla data standard del 01/01/1970 alle 00:00:00 GMT.
  prop.setStartTime("1027348200");
  prop.setEndTime("1027349100");
```

Esempio

Il seguente esempio mostra come è possibile utilizzare molte delle classi commtrace, incluse le classi Format e FormatProperties, per visualizzare le informazioni di traccia nel monitor:

"Esempio: utilizzo delle classi commtrace" a pagina 188

Documentazione di riferimento Javadoc

Per ulteriori informazioni sulle classi Format e FormatProperties, consultare la seguente documentazione di riferimento Javadoc:

Format

FormatProperties

Esecuzione di Format come un programma autonomo:

Oltre ad utilizzare la classe Format nel programma Java, è possibile eseguirla come programma di utilità autonomo, della riga comandi per formattare una traccia di comunicazioni. Il programma collega un IFSFileOutputStream all'outfile specificato e scrive i dati su tale file.

L'esecuzione di Format come programma di utilità autonomo consente all'utente di formattare file utilizzando la potenza di elaborazione e lo spazio di memoria del server iSeries.

Esecuzione di Format da una riga comandi

Per eseguire il programma di utilità Format da una richiesta della riga comandi, utilizzare il seguente comando:

```
java com.ibm.as400.commtrace.Format [options]
```

dove [options] equivale ad una o più delle opzioni disponibili. Le opzioni includono:

- Il sistema a cui ci si vuole collegare

- L'ID utente e la parola d'ordine per il sistema
- La traccia delle comunicazioni che di desidera analizzare
- Il file in cui memorizzare i risultati

Per un elenco completo di opzioni disponibili, consultare le seguenti informazioni:

Documentazione di riferimento Javadoc per la classe Format

Esecuzione di Format in remoto

Per eseguire questa classe in remoto, utilizzare la classe JavaApplicationCall:

```
// Creare un oggetto JavaApplicationCall.
jaCall = new JavaApplicationCall(sys);
// Impostare l'applicazione Java che si desidera eseguire.
jaCall.setJavaApplication("com.ibm.as400.util.commtrace.Format");
// Impostare la variabile di ambiente classpath utilizzata dalla JVM sul
// server, in modo che possa individuare la classe da eseguire.
jaCall.setClassPath("/QIBM/ProdData/OS400/JT400/lib/JT400Native.jar");

String[] args2 =
{ "-c", "true", "-t", "/path/to/trace", "-o", "/path/to/trace.extension"};

jaCall.setParameters(args2);

if (jaCall.run() != true) {
  // Chiamata non riuscita
}
```

Classe Prolog

La classe Prolog rappresenta la sezione iniziale a 256 byte di una traccia delle comunicazioni per una descrizione di linea LAN . Il Prolog contiene informazioni generali sulla traccia, come ad esempio le ore di avvio e di fine, il numero di byte raccolti e così via. Utilizzare la classe Prolog per richiamare informazioni da questa sezione di traccia, che è possibile poi stampare, visualizzare, filtrare o elaborare in altri modi.

La classe Prolog fornisce metodi che consentono di eseguire una gamma di azioni che includono le seguenti:

- Richiamare valori dai campi del prolog, come ad esempio la descrizione di traccia, il tipo Ethernet, la direzione dei dati, l'indirizzo IP e così via
- Restituire una Stringa formattata che contiene tutti i campi del prolog
- Verificare i campi del prolog per i dati non validi

Esempio

Il seguente esempio mostra come è possibile utilizzare molte delle classi commtrace, inclusa la classe Prolog, per visualizzare informazioni di traccia nel monitor:

"Esempio: utilizzo delle classi commtrace" a pagina 188

Documentazione di riferimento Javadoc

Per ulteriori informazioni sulla classe Prolog, consultare la seguente documentazione di riferimento Javadoc:

Prolog

Classe Frame

La classe Frame rappresenta tutti i dati in un record o in una frame, in una traccia delle comunicazioni per una descrizione di linea LAN. Ogni Frame contiene tre sezioni principali di dati che vengono visualizzate nel seguente ordine:

1. Una sezione iniziale a 24 byte che contiene informazioni generali sulla frame
2. Informazioni generali sulla frame (rappresentate dalla classe LanHeader)
3. I dati del pacchetto (rappresentato dalle sottoclassi della classe astratta IPacket)

Utilizzare la classe Frame per analizzare e creare una rappresentazione stampabile dei dati nella frame. La classe Frame conserva i dati del pacchetto in una struttura tipo elenco collegata che utilizza formati specifici. Per informazioni specifiche sui possibili formati per i dati del pacchetto in una frame e per informazioni generali sulla struttura di una frame, consultare i seguenti argomenti:

Documentazione di riferimento Javadoc Frame

Modello Commtrace

La classe Frame fornisce metodi che consentono di eseguire una gamma di azioni che includono le seguenti:

- Richiamare il pacchetto dati
- Richiamare il numero, lo stato ed il tipo della frame
- Restituire dati specifici dalla frame come Stringa formattata

E' possibile utilizzare la seguente procedura per accedere ai dati in un pacchetto:

1. Utilizzare Frame.getPacket() per richiamare il pacchetto
2. Accedere ai dati nell'intestazione richiamando Packet.getHeader()
3. Dopo aver richiamato l'intestazione, richiamare Header.getType() per individuare il tipo
4. Utilizzare la sottoclasse Header specifica per accedere ai dati associati a tale intestazione (il payload) e qualsiasi ulteriore intestazione

Esempio

Il seguente esempio mostra come è possibile utilizzare molte delle classi commtrace, incluse le classi Format e FormatProperties, per visualizzare le informazioni di traccia nel monitor:

"Esempio: utilizzo delle classi commtrace" a pagina 188

Classe LanHeader

La classe LanHeader rappresenta la sezione di dati nella frame che si presenta tra la sezione iniziale di informazioni a 24 byte ed i dati del pacchetto. Questi dati contengono informazioni generali sulla frame.

Utilizzare la classe LanHeader per analizzare e stampare le informazioni nella LanHeader. Il tipo di informazioni contenute dalla LanHeader include:

- Il byte che identifica l'avvio della prima intestazione in questo pacchetto
- Indirizzi MAC (Medium Access Control)
- Indirizzi token ring ed informazioni di instradamento

LanHeader fornisce inoltre due metodi per consentire all'utente di restituire una Stringa formattata che contiene quanto segue:

- Dati di instradamento token ring

- Indirizzi MAC origine, indirizzi MAC destinazione, formato frame e tipo frame

Documentazione di riferimento Javadoc

Per ulteriori informazioni sulla classe LanHeader, consultare la seguente documentazione di riferimento Javadoc:

LanHeader

Classe IPPacket

La classe IPPacket è una superclasse astratta per la creazione di classi che rappresentano tipi specifici di pacchetti. Le sottoclassi di IPPacket includono:
- ARPPacket
- IP4Packet
- IP6Packet
- UnknownPacket

Le classi Packet consentono all'utente di richiamare il tipo di pacchetto ed accedere ai dati grezzi (l'intestazione ed il payload) che il pacchetto contiene. Tutte le sottoclassi utilizzano programmi di creazione simili ed includono un metodo supplementare che restituisce una versione stampabile del contenuto del pacchetto come Stringa.

Tutti i programmi di creazione della classe Packet prendono una schiera di byte dei dati del pacchetto come argomento, ma ARPPacket richiede anche un numero intero che specifica il tipo di frame. La creazione di un'istanza di una classe Packet crea automaticamente l'oggetto Header appropriato.

Le classi Packet forniscono metodi che consentono di eseguire una gamma di azioni che includono le seguenti:
- Richiamare il nome ed il tipo di pacchetto
- Impostare il tipo di pacchetto
- Restituire l'oggetto Header di massimo livello associato al pacchetto
- Restituire tutti i dati del pacchetto come Stringa non formattata
- Restituire dati specifici dal pacchetto come Stringa formattata

Documentazione di riferimento Javadoc

Per ulteriori informazioni sulle classi Packet, consultare la seguente documentazione di riferimento Javadoc:

IPPacket

ARPPacket

IP4Packet

IP6Packet

UnknownPacket

Classe Header

La classe Header è la superclasse astratta per la creazione di classi che rappresentano tipi specifici di intestazioni di pacchetto. Le intestazioni di pacchetto includono i dati associati (o payload), che possono essere altre intestazioni e payload. Le sottoclassi di Header includono:

- ARPHeader
- ExtHeader
- ICMP4Header
- ICMP6Header
- IP4Header
- IP6Header
- TCPHeader
- UDPHeader
- UnknownHeader

Le classi Header consentono di recuperare i dati per l'intestazione e il payload. Una intestazione può incorporare altre intestazioni e i rispettivi payload.

La creazione di un'istanza di una classe Packet crea automaticamente l'oggetto Header appropriato. Le classi Header forniscono metodi che consentono di eseguire una gamma di azioni che includono le seguenti:

- Restituire la lunghezza, il nome e il tipo dell'intestazione
- Richiamare i dati nell'intestazione come una schiera di byte
- Richiamare la successiva intestazione nel pacchetto
- Richiamare il payload come schiera di byte, stringa ASCII e stringa esadecimale
- Restituire tutti i dati dell'intestazione come stringa non formattata
- Restituire dati specifici dall'intestazione come Stringa formattata

Documentazione di riferimento Javadoc

Per ulteriori informazioni sulle classi Header, consultare la seguente documentazione di riferimento Javadoc:

Intestazione

ARPHeader

ExtHeader

ICMP4Header

ICMP6Header

IP4Header

IP6Header

TCPHeader

UDPHeader

UnknownHeader

Esempio: utilizzo delle classi commtrace

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
////////////////////////////////////////////////////////////////////
//
// Esempio che utilizza le classi commtrace per stampare dati della traccia delle
// comunicazioni su un monitor utilizzando un file binario di traccia delle comunicazioni come
// sorgente per i dati.
//
// Sintassi del comando:
//    java CommTraceExample
//
////////////////////////////////////////////////////////////////////

import com.ibm.as400.util.commtrace.*;


public class CommTraceExample {


  public CommTraceExample() {
    // Creare una FormatProperties. Per impostazione predefinita visualizza tutti i dati.
    FormatProperties fmtprop = new FormatProperties();

    Format fmt = new Format("/path/to/file");
    // Imposta le proprietà del filtro per questo formato
    fmt.setFilterProperties(fmtprop);
    fmt.formatProlog(); // Formattare il prolog

    Prolog pro = fmt.getProlog();
    System.out.println(pro.toString());

    // Se questa non è una traccia valida
    if (!pro.invalidData()) {
      Frame rec;

      // Richiamare i record
      while ((rec = fmt.getNextRecord()) != null) {

        // Stampare il numero frame
        System.out.print("Record:" + rec.getRecNum());
        // Stampare l'ora
        System.out.println(" Time:" + rec.getTime());
        // Richiamare questo pacchetto di record
        IPPacket p = rec.getPacket();
        // Richiamare la prima intestazione
        Header h = p.getHeader();

        // Se IPPacket IP6
        if (p.getType() == IPPacket.IP6) {

          // Se Intestazione IP6
          if (h.getType() == Header.IP6) {

            // Convertire in IP6 in modo che sia possibile accedere ai metodi
            IP6Header ip6 = (IP6Header) h;

            System.out.println(h.getName() + " src:" + ip6.getSrcAddr() + " dst:" + ip6.getDstAddr());
            // Stampare l'intestazione come hex
            System.out.println(ip6.printHexHeader());
            // Stampare una rappresentazione di stringa dell'intestazione.
            System.out.println("Complete " + h.getName() + ":\n" + ip6.toString(fmtprop));

            // Richiamare le restanti intestazioni
            while ((h = h.getNextHeader()) != null) {
```

```
              // Se è un'intestazione TCP
              if (h.getType() == Header.TCP) {
                // Convertire in modo che sia possibile accedere ai metodi
                TCPHeader tcp = (TCPHeader) h;
                System.out.println(h.getName() +" src:" + tcp.getSrcPort() +" dst:" + tcp.getDstPort());
                System.out.println("Complete " + h.getName() + ":\n" + tcp.toString(fmtprop));

                // Se è un'intestazione UDP
              } else if (h.getType() == Header.UDP) {
                // Convertire in modo che sia possibile accedere ai metodi
                UDPHeader udp = (UDPHeader) h;
                System.out.println(h.getName() +" src:" + udp.getSrcPort() + " dst:" + udp.getDstPort());
                System.out.println("Complete " + h.getName() + ":\n" + udp.toString(fmtprop));
              }
            }
          }
        }
      }
    }
  }

        public static void main(String[]args) {
        CommTraceExample e = new CommTraceExample();
    }
}
```

Classi HTML

Le classi HTML IBM Toolbox per Java aiutano l'utente a:

- Impostare moduli e tabelle per le pagine HTML
- Allineare il testo
- Gestire varie tag HTML
- Creare i dati sorgente FO (formatting object) XLS (Extensible Stylesheet Language)
- Modificare il linguaggio e la direzione del testo
- Creare elenchi ordinati e non ordinati
- Creare elenchi di file e alberi gerarchici HTML (e gli elementi in essi contenuti)
- Aggiungere attributi tag non ancora definiti nelle classi HTML (ad esempio, gli attributi bgcolor e style)

Le classi HTML realizzano l'interfaccia HTMLTagElement.Ogni classe produce una tag HTML per un tipo di elemento specifico. La tag può essere richiamata utilizzando il metodo getTag(), quindi può essere incorporata in un documento HTML.Le tag create con le classi HTML sono coerenti con la specifica HTML 3.2.

Le classi HTML possono utilizzare le classi servlet per richiamare i dati dal server iSeries.Tuttavia, possono anche essere utilizzate singolarmente se si dispone dei dati della tabella o del modulo.

Inoltre, è possibile utilizzare la classe HTMLDocument per creare facilmente le pagine HTML o i dati sorgente FO XSL. E' possibile convertire i dati FO XSL in documenti PDF (Portable Document Format). Utilizzando il formato PDF i documenti potranno conservare lo stesso aspetto grafico quando vengono stampati e visualizzati elettronicamente.

Le classi HTML facilitano la creazione di moduli, tabelle e altri elementi HTML:

- La classe BidiOrdering consente di modificare la lingua e la direzione del testo.
- La classe DirFilter consente di stabilire se un oggetto file è un indirizzario.
- La classe HTMLAlign consente di allineare blocchi di emissioni HTML.
- La classe HTMLDocument consente di creare più facilmente i dati sorgente HTML o FO XSL.

- La classe HTMLFileFilter consente di stabilire se un oggetto File è un file.
- Le classi HTMLForm aiutano a creare moduli più facilmente rispetto allo script CGI.
- La classe HTMLHead consente di creare tag iniziali per la pagine HTML.
- La classe HTMLHeading consente di creare tag di intestazione per le pagine HTML.
- La classe HTMLHyperlink aiuta a creare collegamenti all'interno di pagine HTML.
- La classe HTMLImage consente di creare le tag immagine per le pagine HTML.
- Le classi HTMLList aiutano a creare degli elenchi per le pagine HTML.
- La classe HTMLMeta consente di creare tag meta per le pagine HTML.
- La classe HTMLParameter specifica i parametri disponibili su HTMLServlet.
- La classe HTMLServlet consente di creare un lato server.
- Le classi HTMLTable aiutano a creare delle tabelle per le pagine HTML.
- La classe HTMLText consente di accedere alle proprietà dei font all'interno delle pagine HTML.
- Le classi HTMLTree consentono di visualizzare un albero gerarchico HTML di elementi HTML.
- La classe URLEncoder codifica i delimitatori da utilizzare in una stringa URL.
- La classe URLParser consente di analizzare una stringa URL per URI, proprietà e riferimenti.

Nota: il file jt400Servlet.jar include la classe HTML e la classe Servlet. E' necessario aggiornare CLASSPATH per puntare al file jt400Servlet.jar se si desidera utilizzare le classi nel pacchetto com.ibm.as400.util.html.

Classe BidiOrdering

La classe BidiOrdering rappresenta una tag HTML che modifica il linguaggio e la direzione del testo.Una stringa <BDO> HTML richiede due attributi, uno per la lingua e l'altro per la direzione del testo.

La classe BidiOrdering consente di:
- Richiamare e impostare l'attributo lingua
- Richiamare e impostare la direzione del testo

Per ulteriori informazioni sull'utilizzo della tag <BDO> HTML, consultare il sito web W3C  .

Esempio: utilizzo di BidiOrdering

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per informazioni legali importanti.

L'esempio che segue crea un oggetto BidiOrdering e imposta il suo linguaggio e direzione:

```
         // Creare un oggetto BidiOrdering ed impostare la lingua e la direzione.
         BidiOrdering bdo = new BidiOrdering();
         bdo.setDirection(HTMLConstants.RTL);
         bdo.setLanguage("AR");


         // Creare del testo.
    HTMLText text = new HTMLText("Some Arabic Text.");
              text.setBold(true);

         // Aggiungere il testo al BidiOrdering e richiamare la tag HTML.
         bdo.addItem(text);
         bdo.getTag();
```

La specifica di stampa produce la tag che segue:

```
<bdo lang="AR" dir="rtl">
  <b>Some Arabic Text.</b>
</bdo>
```

Quando si utilizza questa tag in una pagina HTML, i browser che riconoscono la tag <BDO> visualizzano l'esempio come segue:

.txeT cibarA emoS

Classe HTMLAlign

La classe HTMLAlign consente di allineare sezioni del documento HTML, invece di allineare soltanto elementi singoli, quali paragrafi o intestazioni.

La classe HTMLAlign rappresenta la tag <DIV> e i relativi attributi di allineamento associati. E' possibile utilizzare l'allineamento a destra, a sinistra o al centro.

E' possibile utilizzare questa classe per eseguire una serie di operazioni che includono:

- Aggiungere o eliminare elementi dall'elenco di tag che si desidera allineare
- Richiamare e impostare l'allineamento
- Richiamare e impostare la direzione dell'interpretazione del testo
- Richiamare e impostare la lingua della voce immessa
- Richiamare una rappresentazione di stringa dell'oggetto HTMLAlign

Esempio: creazione degli oggetti HTMLAlign

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per informazioni legali importanti.

L'esempio che segue crea un elenco disordinato, quindi crea un oggetto HTMLAlign per allinearlo:

```
// Creare un elenco non ordinato.
UnorderedList uList = new UnorderedList();
uList.setType(HTMLConstants.DISC);
UnorderedListItem uListItem1 = new UnorderedListItem();
uListItem1.setItemData(new HTMLText("Centered unordered list"));
uList.addListItem(uListItem1);
UnorderedListItem uListItem2 = new UnorderedListItem();
uListItem2.setItemData(new HTMLText("Another item"));
uList.addListItem(uListItem2);

// Allineare l'elenco.
HTMLAlign align = new HTMLAlign(uList, HTMLConstants.CENTER);
System.out.println(align);
```

Il precedente esempio produce la seguente tag:

```
<div align="center">
<ul type="disc">
  <li>Centered unordered list</li>
  <li>Another item</li>
</ul>
```

Quando si utilizza questa tag in una pagina HTML, essa viene visualizzata in questo modo:

- Elenco non ordinato centrato
- Un altro elemento

Classe HTMLDocument

La classe HTMLDocument consente di utilizzare più facilmente classi HTML di IBM Toolbox per Java per creare sia pagine HTML che documenti PDF (Portable Document Format). Quando si crea una classe HTMLDocument, è necessario specificare se quest'ultima contiene tag HTML o tag FO XSL:

- Quando si desidera creare pagine HTML, la classe HTMLDocument consente di raggruppare facilmente tutte le tag HTML necessarie. Tuttavia, le pagine HTML non hanno sempre lo stesso aspetto quando si stampano e si visualizzano in un browser web.
- Quando si desidera creare documenti PDF, la classe HTMLDocument consente di creare il sorgente FO XSL che contiene tutte le informazioni necessarie per produrre il documento PDF. I documenti PDF conservano lo stesso aspetto grafico quando vengono stampati e visualizzati elettronicamente.

Per utilizzare HTMLDocument, è necessario includere un programma di analisi XML e un processore XSLT nella variabile di ambiente CLASSPATH. Per ulteriori informazioni, consultare le seguenti pagine:

"File Jar" a pagina 14

"Programma di analisi XML e processore XSLT" a pagina 417

E' possibile elaborare i dati sorgente HTML o XSL come desiderato, ad esempio, visualizzando l'HTML, salvando l'XSL su un file o utilizzando i dati di flusso in un'altra parte del programma Java.

Per ulteriori informazioni sulla creazione delle pagine HTML e dei dati sorgente FO XSL, consultare le pagine seguenti:
- "Utilizzo di HTMLDocument per creare dati HTML"
- "Utilizzo di HTMLDocument per creare dati FO XSL" a pagina 193
- "Esempi: utilizzo di HTMLDocument" a pagina 195

Documentazione di riferimento Javadoc

Per ulteriori informazioni sulla classe HTMLDocument, consultare la seguente documentazione di riferimento Javadoc:

HTMLDocument

Utilizzo di HTMLDocument per creare dati HTML:

Un HTMLDocument funge da wrapper che conserva le informazioni necessarie per creare sia dati sorgente FO XSL che HTML. Quando si desidera creare pagine HTML, la classe HTMLDocument consente di raggruppare facilmente tutte le tag HTML necessarie.

Creazione di dati sorgente HTML

Quando si crea il sorgente HTML, HTMLDocument richiama le tag HTML dagli oggetti HTML creati. E' possibile creare sia HTMLDocument.getTag() inserire nel flusso tutti gli elementi definiti che getTag() per ogni oggetto HTML individuale.

HTMLDocument crea i dati HTML non appena viene definito nel programma Java, quindi assicurarsi che l'HTML risultante sia completo e corretto.

Quando si richiama HTMLDocument.getTag(), l'oggetto HTMLDocument esegue le azioni riportate di seguito:
- Crea la tag di apertura <HTML>. Alla fine dei dati, crea la tag di chiusura </HTML>.
- Converte gli oggetti HTMLHead e HTMLMeta in tag HTML.
- Crea la tag di apertura <BODY> subito dopo la tag <HEAD>. Alla fine dei dati, prima della tag di chiusura </HTML>, crea la tag di chiusura </BODY>.

  Nota: se non viene specificata una tag <HEAD>, HTMLDocument crea la tag <BODY> dopo la tag <HTML>.

- Converte gli oggetti HTML rimanenti in tag HTML come richiesto dal programma.

Nota: HTMLDocument inserisce nel flusso le tag HTML come richiesto dal programma Java, quindi assicurarsi di richiamare le tag nell'ordine appropriato.

Esempi: utilizzo di HTMLDocument

Il seguente esempio mostra come utilizzare HTMLDocument per creare i dati sorgente HTML (e il sorgente FO XSL):

"Esempio: utilizzo di HTMLDocument per creare entrambi i sorgenti HTML e FO XSL" a pagina 198

Documentazione di riferimento Javadoc

Per ulteriori informazioni sulla classe HTMLDocument, consultare la seguente documentazione di riferimento Javadoc:

HTMLDocument

Il seguente esonero di responsabilità si applica a tutti gli esempi IBM Toolbox per Java:

Utilizzo di HTMLDocument per creare dati FO XSL:

Un HTMLDocument funge da wrapper che conserva le informazioni necessarie per creare sia dati sorgente FO XSL che HTML. Il sorgente FO XSL creato segue il modello di formattazione FO XSL. Il modello utilizza elementi rettangolari, denominati aree, per conservare elementi del contenuto individuali, quali immagini, testo, altri FO XSL o niente. Il seguente elenco descrive i quattro tipi di aree di base:

- Le regioni fungono da contenitore di livello più elevato.
- Le aree di blocco rappresentano elementi di livello di blocco, ad esempio, paragrafi o voci di elenco.
- Le aree di riga rappresentano una riga di testo all'interno di un blocco.
- Le aree interne alla riga rappresentano parti di una riga, ad esempio, un singolo carattere, un piè di pagina o un'equazione matematica.

Le tag FO XSL create dall'IBM Toolbox per Java sono conforme agli standard XSL descritti nelle raccomandazioni W3C. Per ulteriori informazioni su XSL, FO XSL e sulle raccomandazioni W3C, consultare il seguente sito web:

Extensible Stylesheet Language (XSL) Versione 1.0

Creazione di dati sorgente FO XSL

Quando si crea il sorgente FO XSL, le proprietà HTMLDocument rappresentano le tag FO XSL che specificano la dimensione della pagina, l'orientamento e i margini. Inoltre, HTMLDocument recupera da molte classi HTML le tag FO XSL corrispondenti per tale elemento del contenuto.

Dopo aver utilizzato HTMLDocument per creare il sorgente FO XSL, è possibile utilizzare un programma di formattazione XSL (ad esempio, la classe XSLReportWriter) per posizionare gli elementi del contenuto nelle pagine di un documento.

HTMLDocument crea i dati sorgente FO XSL in due sezioni principali:

- La prima sezione contiene le tag FO XSL <fo:root> e <fo:layout-master-set> che dispongono delle informazioni sul layout di pagina generali relative all'altezza, all'ampiezza e ai margini della pagina. Per specificare i valori per le informazioni sul layout, utilizzare i metodi di impostazione HTMLDocument per impostare i valori per le proprietà associate.
- La seconda sezione contiene la tag <fo:page-sequence> di FO XSL che dispone degli elementi del contenuto individuali. Per specificare gli elementi del contenuto individuali, quali istanze delle classi HTML, richiamare la tag FO XSL corrispondente dall'oggetto HTML. Assicurarsi che per l'elemento del contenuto vengano utilizzate solo classi HTML che dispongano del metodo getFoTag().

  Nota: il tentativo di richiamare le tag FO XSL dalle classi HTML che non dispongono del metodo getFoTag() darà origine a una tag di commento.

Per ulteriori informazioni sulle classi HTML che comprendono i metodi per la gestione delle tag FO XSL, consultare la seguente documentazione di riferimento Javadoc:

"Classi abilitate a FO XSL" a pagina 195

Dopo aver creato un'istanza di HTMLDocument e impostato le proprietà di layout, richiamare le tag FO XSL dagli oggetti HTML utilizzando i metodi setUseFO(), getFoTag() e getTag().

- E' possibile utilizzare setUseFO() sia in HTMLDocument che negli oggetti individuali HTML. Quando si utilizza setUseFO(), è possibile richiamare le tag FO XSL utilizzando HTMLDocument.getTag().
- In alternativa, è possibile utilizzare il metodo getFoTag() sia in HTMLDocument che negli oggetti individuali HTML. E' possibile utilizzare questa metodo alternativo quando si desidera creare entrambi i sorgenti FO XSL e HTML da HTMLDocument o dagli oggetti HTML.

Esempio: utilizzo di HTMLDocument

Dopo aver creato i dati sorgente FO XSL, è necessario convertire tali dati in un formato che gli utenti potranno visualizzare e stampare. I seguenti esempi, mostrano come creare i dati sorgente FO XSL (e sorgente HTML) e convertirli in un documento PDF utilizzando le classi XSLReportWriter e Context:

"Esempio: utilizzo di HTMLDocument per creare entrambi i sorgenti HTML e FO XSL" a pagina 198

"Esempio: conversione di dati sorgente FO XSL in PDF" a pagina 196

Documentazione di riferimento Javadoc

Per ulteriori informazioni sulla classe HTMLDocument, consultare la seguente documentazione di riferimento Javadoc:

HTMLDocument

Il seguente esonero di responsabilità si applica a tutti gli esempi IBM Toolbox per Java:

Esonero di responsabilità per gli esempi di codice

IBM fornisce una licenza non esclusiva per utilizzare ciò come esempio da cui creare funzioni simili personalizzate, in base a richieste specifiche.

Tutti gli esempi di codice forniti dall'IBM hanno la sola funzione illustrativa. Questi esempi non sono stati interamente testati in tutte le condizioni. IBM, perciò, non fornisce nessun tipo di garanzia o affidabilità implicita, rispetto alla funzionalità o alle funzioni di questi programmi.

Tutti i programmi qui contenuti vengono forniti all'utente ″COSI' COME SONO″ senza garanzie di alcun tipo. Le garanzie implicite di non contraffazione, commerciabilità e adeguatezza a scopi specifici sono espressamente vietate.

Classi abilitate a FO XSL:

Molte classi HTML di IBM Toolbox per Java utilizzano i seguenti metodi, i quali abilitano le istanze di tali classi a gestire HTMLDocument:

- getFoTag()
- getTag()
- setUseFO()

Per ulteriori informazioni sulla classe HTMLDocument e sulle classi HTML che includono i metodi per la gestione dei FO XSL, consultare la seguente documentazione di riferimento Javadoc:

- HTMLDocument
- BidiOrdering
- HTMLAlign
- HTMLHead
- HTMLHeading
- HTMLImage
- HTMLList
- HTMLListItem
- HTMLTable
- HTMLTableCaption
- HTMLTableCell
- HTMLTableHeader
- HTMLTableRow
- HTMLTagElement
- OrderedList
- UnorderedList

Esempi: utilizzo di HTMLDocument:

I seguenti esempi mostrano come utilizzare la classe HTMLDocument per creare i dati sorgente FO XSL.

Esempio: utilizzo di HTMLDocument per creare entrambi i sorgenti HTML e FO XSL

Il seguente esempio mostra come creare entrambi i dati sorgente HTML e FO XSL contemporaneamente:

"Esempio: utilizzo di HTMLDocument per creare entrambi i sorgenti HTML e FO XSL" a pagina 198

Esempio: conversione di dati sorgente FO XSL in PDF

Dopo aver creato i dati sorgente FO XSL, è necessario convertire tali dati in un formato che gli utenti potranno visualizzare e stampare. Il seguente esempio mostra come convertire un file che contiene i dati sorgente FO XSL in un documento PDF utilizzando le classi XSLReportWriter e Context:

"Esempio: conversione di dati sorgente FO XSL in PDF"

Il seguente esonero di responsabilità si applica a tutti gli esempi IBM Toolbox per Java:

Esonero di responsabilità per gli esempi di codice

IBM fornisce una licenza non esclusiva per utilizzare ciò come esempio da cui creare funzioni simili personalizzate, in base a richieste specifiche.

Tutti gli esempi di codice forniti dall'IBM hanno la sola funzione illustrativa. Questi esempi non sono stati interamente testati in tutte le condizioni. L'IBM, perciò, non intende implicita alcuna garanzia di affidabilità, manutenibilità o funzionalità di questi programmi.

Tutti i programmi qui contenuti sono forniti ″COSI' COME SONO″ senza garanzie di alcun tipo. Sono espressamente smentite tutte le garanzie implicite di non violazione, di commerciabilità e idoneità per scopi specifici.

Il seguente esonero di responsabilità si applica a tutti gli esempi IBM Toolbox per Java:

Esonero di responsabilità per gli esempi di codice

IBM fornisce una licenza non esclusiva per utilizzare ciò come esempio da cui creare funzioni simili personalizzate, in base a richieste specifiche.

Tutti gli esempi di codice forniti dall'IBM hanno la sola funzione illustrativa. Questi esempi non sono stati interamente testati in tutte le condizioni. L'IBM, perciò, non intende implicita alcuna garanzia di affidabilità, manutenibilità o funzionalità di questi programmi.

Tutti i programmi qui contenuti sono forniti ″COSI' COME SONO″ senza garanzie di alcun tipo. Sono espressamente smentite tutte le garanzie implicite di non violazione, di commerciabilità e idoneità per scopi specifici.

Esempio: conversione di dati sorgente FO XSL in PDF:

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per informazioni legali importanti.

```
////////////////////////////////////////////////////////////////////////////
//
// Esempio: conversione di dati sorgente FO XSL in PDF.
//
// Questo programma utilizza le classi ReportWriter di IBM Toolbox per Java per convertire
// i dati sorgente FO XSL (creati tramite l'utilizzo di HTMLDocument) in PDF.
//
// In questo esempio, i seguenti file .jars devono trovarsi in un classpath.
//
//   composer.jar
//   outputwriters.jar
//   reportwriter.jar
//   x4j400.jar
//   xslparser.jar
//
// Tali file jars sono parte dell'IBM ToolBox per Java e risiedono nell'indirizzario
// /QIBM/ProdData/HTTP/Public/jt400/lib sul server.
//
```

```
// Sintassi del comando:
//    ProcessXslFo FOfilename PDFfilename
//
////////////////////////////////////////////////////////////////////////////


import java.io.FileInputStream;
import java.io.FileOutputStream;

import java.awt.print.Paper;
import java.awt.print.PageFormat;

import org.w3c.dom.Document;

import com.ibm.xsl.composer.framework.Context;

import com.ibm.as400.util.reportwriter.pdfwriter.PDFContext;
import com.ibm.as400.util.reportwriter.processor.XSLReportProcessor;

public class ProcessXslFo
{
        public static void main(String args[])
    {
            if (args.length != 2)
        {
            System.out.println("Usage:  java ProcessXslFo <fo file name> <pdf file name>");
                    System.exit(0);
        }

                                try
        {
            String inName = args[0];
            String outName = args[1];

            /* Input. File contenente l'FO XML. */
            FileInputStream fin = null;

            /* Output. Che in questo esempio sarà un PDF. */
            FileOutputStream fout = null;

                                    try
            {
                fin  = new FileInputStream(inName);
                fout = new FileOutputStream(outName);
            }
                                    catch (Exception e)
            {
                        e.printStackTrace();
                        System.exit(0);
            }

            /*
            * Setup Page format.
            */
                    Paper paper = new Paper();
            paper.setSize(612, 792);
            paper.setImageableArea(0, 0, 756, 936);

            PageFormat pageFormat = new PageFormat();
            pageFormat.setPaper(paper);

            /*
            * Create a PDF context. Set output file name.
            */
            PDFContext pdfContext = new PDFContext(fout, pageFormat);

            /*
```

```
            * Create XSLReportProcessor instance.
            */
            XSLReportProcessor report = new XSLReportProcessor(pdfContext);

            /*
            * Open XML FO source.
            */
                                        try
            {
                report.setXSLFOSource(fin);
            }
                                        catch (Exception e)
            {
                        e.printStackTrace();
                        System.exit(0);
            }

            /*
            * Process the report.
            */
                                        try
            {
                report.processReport();
            }
                                        catch (Exception e)
            {
                        e.printStackTrace();
                        System.exit(0);
            }
        }
                                    catch (Exception e)
        {
                    e.printStackTrace();
                    System.exit(0);
        }

        /* exit */
                System.exit(0);
    }
}
```

Esempio: utilizzo di HTMLDocument per creare entrambi i sorgenti HTML e FO XSL:

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per informazioni legali importanti.

```
////////////////////////////////////////////////////////////////////////////
//
// Esempio: utilizzo della classe HTMLDocument di Toolbox
// per creare entrambi i dati sorgente HTML e FO XSL.
//
// Questo programma utilizza la classe HTMLDocument
// per creare due file: uno che disponga del sorgente HTML e
// l'altro che disponga del sorgente FO XSL.
//
// Sintassi del comando:
//    HTMLDocumentExample
//
////////////////////////////////////////////////////////////////////////////


import com.ibm.as400.util.html.*;
import java.*;
import java.io.*;
import java.lang.*;
import java.beans.PropertyVetoException;

public class HTMLDocumentExample
```

```
{
        public static void main (String[] args)
    {
        //Creare la classe HTMLDocument che conservi le proprietà documento necessarie
        HTMLDocument doc = new HTMLDocument();

        //Impostare le proprietà della pagina e del margine.  I numeri sono espressi in pollici.
        doc.setPageWidth(8.5);
        doc.setPageHeight(11);
        doc.setMarginTop(1);
        doc.setMarginBottom(1);
        doc.setMarginLeft(1);
        doc.setMarginRight(1);

        //Creare un'intestazione per la pagina.
        HTMLHead head = new HTMLHead();
        //Impostare il titolo per l'intestazione
        head.setTitle("This is the page header.");

        //Creare diverse intestazioni
        HTMLHeading h1 = new HTMLHeading(1, "Heading 1");
        HTMLHeading h2 = new HTMLHeading(2, "Heading 2");
        HTMLHeading h3 = new HTMLHeading(3, "Heading 3");
        HTMLHeading h4 = new HTMLHeading(4, "Heading 4");
        HTMLHeading h5 = new HTMLHeading(5, "Heading 5");
        HTMLHeading h6 = new HTMLHeading(6, "Heading 6");

        //Creare del testo che verrà stampato dal lato destro a quello sinistro.
        //Creare l'oggetto BidiOrdering e impostare la direzione
             BidiOrdering bdo = new BidiOrdering();
             bdo.setDirection(HTMLConstants.RTL);

        //Creare del testo
        HTMLText text = new HTMLText("This is Arabic text.");
        //Aggiungere il testo all'oggetto di ordinamento bidirezionale
             bdo.addItem(text);

                  // Creare un'UnorderedList.
             UnorderedList uList = new UnorderedList(HTMLConstants.SQUARE);
                  // Creare ed impostare i dati per UnorderedListItems.
             UnorderedListItem listItem1 = new UnorderedListItem();
             UnorderedListItem listItem2 = new UnorderedListItem();
             listItem1.setItemData(new HTMLText("First item"));
             listItem2.setItemData(new HTMLText("Second item"));
                  // Aggiungere le voci di elenco all'UnorderedList.
             uList.addListItem(listItem1);
             uList.addListItem(listItem2);

                  // Creare un'OrderedList.
             OrderedList oList = new OrderedList(HTMLConstants.SMALL_ROMAN);
                  // Creare l'OrderedListItems.
        OrderedListItem olistItem1 = new OrderedListItem();
        OrderedListItem olistItem2 = new OrderedListItem();
        OrderedListItem olistItem3 = new OrderedListItem();
                  // Impostare i dati in OrderedListItems.
        olistItem1.setItemData(new HTMLText("First item"));
        olistItem2.setItemData(new HTMLText("Second item"));
        olistItem3.setItemData(new HTMLText("Third item"));
                  // Aggiungere le voci di elenco all'OrderedList.
        oList.addListItem(olistItem1);
        oList.addListItem(olistItem2);
                  // Aggiungere (nidificare) l'elenco non ordinato in OrderedListItem2
             oList.addList(uList);
                  // Aggiungere un'altra OrderedListItem all'OrderedList
                  // dopo l'UnorderedList nidificata.
        oList.addListItem(olistItem3);
```

```
// Creare un oggetto HTMLTable predefinito.
         HTMLTable table = new HTMLTable();
                           try
{
    // Impostare gli attributi di tabella.
    table.setAlignment(HTMLTable.LEFT);
    table.setBorderWidth(1);

    // Creare un oggetto HTMLTableCaption predefinito ed impostare il testo del titolo.
          HTMLTableCaption caption = new HTMLTableCaption();
    caption.setElement("Customer Account Balances - January 1, 2000");

    // Impostare il titolo.
          table.setCaption(caption);

    // Creare le intestazioni di tabella ed aggiungerle alla tabella.
    HTMLTableHeader account_header = new HTMLTableHeader(new HTMLText("ACCOUNT"));
    HTMLTableHeader name_header = new HTMLTableHeader(new HTMLText("NAME"));
    HTMLTableHeader balance_header = new HTMLTableHeader(new HTMLText("BALANCE"));

    table.addColumnHeader(account_header);
    table.addColumnHeader(name_header);
    table.addColumnHeader(balance_header);

    // Aggiunge righe alla tabella. Ogni record del cliente rappresenta una riga nella tabella.
    int numCols = 3;
    for (int rowIndex=0; rowIndex< 5; rowIndex++)
    {
        HTMLTableRow row = new HTMLTableRow();
        row.setHorizontalAlignment(HTMLTableRow.CENTER);

        HTMLText account = new HTMLText("000" + rowIndex);
        HTMLText name = new HTMLText("Customer" + rowIndex);
        HTMLText balance = new HTMLText("" + (rowIndex + 1)*200);

        row.addColumn(new HTMLTableCell(account));
        row.addColumn(new HTMLTableCell(name));
        row.addColumn(new HTMLTableCell(balance));

           // Aggiungere la riga alla tabella.
        table.addRow(row);
    }
}
  catch(Exception e)
{
    System.out.println("Problem creating table");
            System.exit(0);
}

//Aggiungere elementi a HTMLDocument
doc.addElement(head);
doc.addElement(h1);
doc.addElement(h2);
doc.addElement(h3);
doc.addElement(h4);
doc.addElement(h5);
doc.addElement(h6);
doc.addElement(oList);
doc.addElement(table);
doc.addElement(bdo);

//Stampare le tag fo su un file.
                           try
{
    FileOutputStream fout = new FileOutputStream("FOFILE.fo");
    PrintStream pout = new PrintStream(fout);
    pout.println(doc.getFOTag());
```

```
        }
                                    catch (Exception e)
        {
            System.out.println("Unable to write fo tags to FOFILE.fo");
        }

        //Stampare le tag html su un file
                                    try
        {
            FileOutputStream htmlout = new FileOutputStream("HTMLFILE.html");
            PrintStream phtmlout = new PrintStream(htmlout);
            phtmlout.println(doc.getTag());
        }
                                    catch (Exception e)
        {
            System.out.println("Unable to write html tags to HTMLFILE.html");
        }

    }
}
```

Classi HTML Form

La classe HTMLForm rappresenta un modulo HTML.Questa classe consente di:

- Aggiungere un elemento, come un pulsante, un hyperlink o una tabella HTML ad un modulo
- Rimuovere un elemento da un modulo
- Impostare altri attributi del modulo, ad esempio il metodo da utilizzare per inviare il contenuto del modulo al server, l'elenco di parametri nascosti o l'indirizzo URL dell'azione

Il programma di creazione dell'oggetto HTMLForm utilizza un indirizzo URL. Questo indirizzo viene definito come URL di azione. E' l'ubicazione dell'applicazione sul server che elaborerà l'immissione del modulo. L'URL di azione può essere specificata sul programma di creazione o impostando l'indirizzo con il metodo setURL().Gli attributi del modulo vengono impostati utilizzando vari metodi set e richiamati utilizzando vari metodi get.

E' possibile aggiungere un qualsiasi elemento tag HTML ad un oggetto HTMLForm, utilizzando addElement(), mentre è possibile rimuoverlo utilizzando removeElement().Utilizzare le seguenti classi di elementi tag HTML in HTMLForms:

- Classi FormInput: rappresentano elementi di immissione per un modulo HTML
- Classi LayoutFormPanel: rappresentano il layout degli elementi del modulo per un modulo HTML
- TextAreaFormElement: rappresenta un elemento area testo in un modulo HTML
- LabelFormElement: rappresenta un'etichetta per un elemento modulo HTML
- SelectFormElement: rappresenta un tipo di immissione di selezione per un modulo HTML
- SelectOption: rappresenta un'opzione per l'oggetto SelectFormElement in un modulo HTML
- RadioFormInputGroup: rappresenta un gruppo di oggetti di immissione radio che consentono all'utente di selezionare un oggetto da un gruppo

Naturalmente, è possibile aggiungere altri elementi tag ad un modulo, incluso quanto segue:

- HTMLText
- HTMLHyperlink
- HTMLTable

Per ulteriori informazioni sull'utilizzo della classe HTMLForm per creare un modulo, consultare questo esempio e l'emissione risultante.

Classi FormInput:

La classe FormInput consente di:

- Richiamare e impostare il nome di un elemento di immissione
- Richiamare e impostare la dimensione di un elemento di immissione
- Richiamare e impostare il valore iniziale di un elemento di immissione

La classe FormInput viene estesa dalle classi nell'elenco seguente. Queste classi consentono di creare tipi specifici di elementi di immissione modulo e consentono di richiamare e impostare vari attributi o richiamare la tag HTML per l'elemento di immissione:

- ButtonFormInput: rappresenta un elemento pulsante per un formato HTML
- FileFormInput: rappresenta un tipo di immissione file, per un formato HTML
- HiddenFormInput: rappresenta un tipo di immissione nascosta per un formato HTML
- ImageFormInput: rappresenta un tipo di immissione immagine per un formato HTML.
- ResetFormInput: rappresenta l'utilizzo del pulsante Ripristina per un formato HTML
- SubmitFormInput: rappresenta l'utilizzo del pulsante Inoltra per un modulo HTML
- TextFormInput: rappresenta una singola riga dell'immissione testo per un formato HTML in cui è possibile definire il numero massimo di caratteri in una riga.Per un tipo di immissione della parola d'ordine è possibile utilizzare PasswordFormInput, che estende TextFormInput e rappresenta un tipo di immissione della parola d'ordine per un modulo HTML
- ToggleFormInput: rappresenta un tipo di immissione alternata per un formato HTML.E' possibile impostare o richiamare l'etichetta del testo e specificare se l'immissione alternata deve essere selezionata o spuntata. Il tipo di immissione alternata può essere:
  - RadioFormInput: rappresenta un tipo di immissione pallino per un formato HTML.I pallini possono essere posizionati in gruppi con la classe RadioFormInputGroup; in questo modo si crea un gruppo di pallini in cui è possibile selezionare solo una delle alternative presentate.
  - CheckboxFormInput: rappresenta un tipo di immissione casella di spunta per un formato HTML in cui è possibile selezionare più di un'alternativa presentata, in cui la casella di spunta viene inizializzata come selezionata o meno.

Classe ButtonFormInput:

La classe ButtonFormInput rappresenta un elemento pulsante per un modulo HTML.

Il seguente esempio mostra come creare un oggetto ButtonFormInput:

```
ButtonFormInput button = new ButtonFormInput("button1", "Press Me", "test()");
    System.out.println(button.getTag());
```

Questo esempio produce la seguente tag:

```
<input type="button" name="button1" value="Press Me" onclick="test()" />
```

Classe FileFormInput:

La classe FileFormInput rappresenta un tipo di file di immissione in un formato HTML.

Il seguente esempio di codice mostra come creare un nuovo oggetto FileFormInput

```
FileFormInput file = new FileFormInput("myFile");
    System.out.println(file.getTag());
```

Il codice precedente crea l'emissione che segue:

```
<input type="file" name="myFile" />
```

Classe HiddenFormInput:

La classe HiddenFormInput rappresenta un tipo di immissione nascosta in un formato HTML.

L'esempio di codice che segue mostra come creare un oggetto HiddenFormInput:

```
        HiddenFormInput hidden = new HiddenFormInput("account", "123456");
        System.out.println(hidden.getTag()):
```

Il codice precedente crea la seguente tag:

```
<input type="hidden" name="account" value="123456" />
```

In una pagina HTML, HiddenInputType non viene visualizzato. Invia le informazioni (in questo caso il numero di conto) nuovamente al server.

Classe ImageFormInput:

La classe ImageFormInput rappresenta un tipo di immissione immagine in un formato HTML.

E' possibile richiamare e aggiornare molti attributi della classe ImageFormInput utilizzando i metodi forniti.

- Richiamare o impostare il sorgente
- Richiamare oimpostare l'allineamento
- Richiamare o impostare l'altezza
- Richiamare o impostare l'ampiezza

Esempio: creazione di un oggetto ImageFormInput

L'esempio di codice che segue mostra come creare un oggetto ImageFormInput:

```
    ImageFormInput image = new ImageFormInput("myPicture", "myPicture.gif");
        image.setAlignment(HTMLConstants.TOP);
        image.setHeight(81);
        image.setWidth(100);
```

L'esempio di codice precedente crea la tag che segue:

```
    <input type="image" name="MyPicture" src="myPicture.gif" align="top" height="81" width="100" />
```

Classe ResetFormInput:

La classe ResetFormInput rappresenta un tipo di immissione del pulsante di ripristino in un formato HTML.

Il seguente esempio di codice mostra come creare un oggetto ResetFormInput:

```
    ResetFormInput reset = new ResetFormInput();
    reset.setValue("Reset");
        System.out.println(reset.getTag());
```

L'esempio di codice precedente crea la seguente tag HTML:

```
    <input type="reset" value="Reset" />
```

Classe SubmitFormInput:

La classe SubmitFormInput rappresenta un tipo di immissione pulsante di inoltro in un formato HTML.

L'esempio codice che segue mostra come creare un oggetto SubmitFormInput:

```
        SubmitFormInput submit = new SubmitFormInput();
    submit.setValue("Send");
        System.out.println(submit.getTag());
```

L'esempio codice precedente crea l'emissione che segue:

```
<input type="submit" value="Send" />
```

Classe TextFormInput:

La classe TextFormInput rappresenta un tipo di immissione di testo a riga singola in formato HTML.La classe TextFormInput fornisce i metodi che consentono di richiamare e impostare il numero massimo di caratteri che un utente può immettere nel campo testo.

Il seguente esempio mostra come creare un nuovo oggetto TextFormInput:

```
TextFormInput text = new TextFormInput("userID");
    text.setSize(40);
    System.out.println(text.getTag());
```

Il precedente esempio di codice crea la seguente tag:

```
<input type="text" name="userID" size="40" />
```

Classe PasswordFormInput:

La classe PasswordFormInput rappresenta un tipo di campo di immissione della parola d'ordine in un formato HTML.

Il seguente esempio di codice mostra le modalità di creazione di un nuovo oggetto PasswordFormInput:

```
PasswordFormInput pwd = new PasswordFormInput("password");
    pwd.setSize(12);
    System.out.println(pwd.getTag());
```

Il precedente esempio di codice crea la seguente tag:

```
<input type="password" name="password" size="12" />
```

Classe RadioFormInput:

La classe RadioFormInput rappresenta un tipo di immissione tramite pallino in un formato HTML. Quando viene creato, il pallino può essere inizializzato come selezionato.

Una serie di pallini con lo stesso nome controllo costruisce un gruppo di pallini. La classe RadioFormInputGroup crea gruppi di pallini.E' possibile selezionare un solo pallino per volta all'interno del gruppo. Inoltre, un pulsante specifico può essere inizializzato come selezionato quando viene creato il gruppo.

Il seguente esempio di codice mostra come creare un oggetto RadioFormInput:

```
RadioFormInput radio = new RadioFormInput("age", "twentysomething", "Age 20 - 29", true);
    System.out.println(radio.getTag());
```

L'esempio di codice precedente crea la tag che segue:

```
<input type="radio" name="age" value="twentysomething" checked="checked" />
```

Classe CheckboxFormInput:

La classe CheckboxFormInput rappresenta un tipo di immissione della casella di spunta in formato HTML. L'utente può selezionare più di una delle opzioni presentate come caselle di spunta in un modulo.

Il seguente esempio mostra come creare un nuovo oggetto CheckboxFormInput:

```
CheckboxFormInput checkbox = new CheckboxFormInput("uscitizen", "yes", "textLabel", true);
    System.out.println(checkbox.getTag());
```

Il codice precedente produce la seguente emissione:

```
<input type="checkbox" name="uscitizen" value="yes" checked="checked" /> textLabel
```

Classe LayoutFormPanel:

La classe LayoutFormPanel rappresenta un layout di elementi del modulo per un modulo HTML. E' possibile utilizzare i metodi forniti da LayoutFormPanel per aggiungere e rimuovere elementi da un pannello o richiamare il numero di elementi nel layout. E' possibile scegliere di utilizzare uno dei due layout:

- "GridLayoutFormPanel": rappresenta un layout a griglia di elementi del modulo per un modulo HTML
- "Classe LineLayoutFormPanel" a pagina 206: rappresenta un layout riga di elementi del modulo per un modulo HTML

GridLayoutFormPanel:

La classe GridLayoutFormPanel rappresenta un layout a griglia degli elementi del modulo.Si utilizza questo layout per un modulo HTML dove si specifica il numero di colonne per la griglia.

L'esempio che segue crea un oggetto GridLayoutFormPanel con due colonne:

```
       // Creare un elemento di immissione modulo testo per il sistema.
LabelFormElement sysPrompt = new LabelFormElement("System:");
TextFormInput system = new TextFormInput("System");

       // Creare un elemento di immissione modulo testo per l'ID utente.
LabelFormElement userPrompt = new LabelFormElement("User:");
TextFormInput user = new TextFormInput("User");

       // Creare un elemento di immissione modulo parola d'ordine per la parola d'ordine.
LabelFormElement passwordPrompt = new LabelFormElement("Password:");
PasswordFormInput password = new PasswordFormInput("Password");

       // Creare l'oggetto GridLayoutFormPanel con due colonne ed aggiungere gli elementi del modulo.
GridLayoutFormPanel panel = new GridLayoutFormPanel(2);
panel.addElement(sysPrompt);
panel.addElement(system);
panel.addElement(userPrompt);
panel.addElement(user);
panel.addElement(passwordPrompt);
panel.addElement(password);

       // Creare il pulsante di inoltro per il modulo.
SubmitFormInput logonButton = new SubmitFormInput("logon", "Logon");

       // Creare l'oggetto HTMLForm ed aggiungere ad esso il pannello.
HTMLForm form = new HTMLForm(servletURI);
form.addElement(panel);
form.addElement(logonButton);
```

Questo esempio crea il seguente codice HTML:

```
<form action=servletURI method="get">
<table border="0">
<tr>
<td>System:</td>
<td><input type="text" name="System" /></td>
</tr>
<tr>
<td>User:</td>
<td><input type="text" name="User" /></td>
</tr>
<tr>
```

```
<td>Password:</td>
<td><input type="password" name="Password" /></td>
</tr>
</table>
<input type="submit" name="logon" value="Logon" />
</form>
```

Classe LineLayoutFormPanel:

La classe LineLayoutFormPanel rappresenta un layout di riga di elementi del modulo per un modulo HTML.Gli elementi del modulo sono sistemati su una singola riga in un pannello.

Esempio: utilizzo di LineLayoutFormPanel

Questo esempio crea un oggetto LineLayoutFormPanel e aggiunge due elementi del modulo.

```
CheckboxFormInput privacyCheckbox =
   new CheckboxFormInput("confidential", "yes", "Confidential", true);
CheckboxFormInput mailCheckbox =
   new CheckboxFormInput("mailingList", "yes", "Join our mailing list", false);
    LineLayoutFormPanel panel = new LineLayoutFormPanel();
    panel.addElement(privacyCheckbox);
    panel.addElement(mailCheckbox);
    String tag = panel.getTag();
```

L'esempio di codice precedente crea il seguente codice HTML:

```
<input type="checkbox" name="confidential" value="yes" checked="checked" /> Confidential
<input type="checkbox" name="mailingList" value="yes" /> Join our mailing list <br/>
```

Classe TextAreaFormElement:

La classe TextAreaFormElement rappresenta un elemento dell'area testo in formato HTML. Si specifica la dimensione dell'area testo impostando il numero di righe e colonne. E' possibile determinare la dimensione impostata per un elemento dell'area testo con i metodigetRows() e getColumns().

Il testo iniziale nell'area testo viene impostato con il metodo setText().Si utilizza il metodogetText() per determinare l'impostazione del testo iniziale.

L'esempio seguente mostra come creare un TextAreaFormElement:

```
TextAreaFormElement textArea = new TextAreaFormElement("foo", 3, 40);
textArea.setText("Default TEXTAREA value goes here");
    System.out.println(textArea.getTag());
```

L'esempio di codice precedente crea il seguente codice HTML:

```
<form>
<textarea name="foo" rows="3" cols="40">
    Default TEXTAREA value goes here
</textarea>
</form>
```

Classe LabelFormElement:

La classe LabelFormElement rappresenta una etichetta per l'elemento formato HTML.La classe LabelFormElement viene utilizzata per etichettare gli elementi del formato HTML come area testo o immissione formato parola d'ordine.L'etichetta è una riga di testo che si imposta utilizzando il metodo setLabel().Questo testo non risponde all'immissione dell'utente e serve per semplificare la comprensione del modulo all'utente.

Esempio: utilizzo di LabelFormElement

Il seguente esempio di codice mostra come creare un oggetto LabelFormElement:

```
LabelFormElement label = new LabelFormElement("Account Balance");
System.out.println(label.getTag());
```

Questo esempio produce le seguenti emissioni:

```
Account Balance
```

Classe SelectFormElement:

La classe SelectFormElement rappresenta un tipo di immissione di selezione per un formato HTML.E' possibileaggiungere e rimuovere varie opzioni all'interno dell'elemento di selezione.

In SelectFormElement vi sono metodi disponibili che consentono di visualizzare e modificare gli attributi dell'elemento di selezione:

- UtilizzaresetMultiple() per stabilire se l'utente può utilizzare o meno più di una opzione
- Utilizzare getOptionCount() per determinare quanti elementi si trovano nel layout dell'opzione
- Utilizzare setSize() per impostare il numero di opzioni visibili all'interno dell'elemento di selezione e utilizzare getSize() per determinare il numero di opzioni visibili.

L'esempio che segue crea un oggetto SelectFormElement con tre opzioni. L'oggetto SelectFormElement denominato *list*, è evidenziato. Le prime due opzioni aggiunte specificano il testo, il nome e gli attributi di selezione dell'opzione. La terza opzione aggiunta viene definita da un oggetto SelectOption.

```
SelectFormElement list = new SelectFormElement("list1");
SelectOption option1 = list.addOption("Option1", "opt1");
SelectOption option2 = list.addOption("Option2", "opt2", false);
SelectOption option3 = new SelectOption("Option3", "opt3", true);
list.addOption(option3);
System.out.println(list.getTag());
```

L'esempio di codice precedente produce il codice HTML che segue:

```
<select name="list1">
<option value="opt1">Option1</option>
<option value="opt2">Option2</option>
<option value="opt3" selected="selected">Option3</option>
</select>
```

Classe SelectOption:

La classe SelectOption rappresenta un'opzione in SelectFormElement HTML. Si utilizza l'elemento del formato di opzione in un formato di selezione.

Sono forniti metodi che è possibile utilizzare per richiamare ed impostare attributi all'interno di una classe SelectOption. Ad esempio, è possibile stabilire se l'opzione è impostata in moda da essere selezionata. E' inoltre possibile impostare il valore di immissione che essa utilizzerà quando il modulo sarà inoltrato.

L'esempio che segue crea tre oggetti SelectOption all'interno di un modulo di selezione. Ognuno degli oggetti SelectOption che seguono sono evidenziati. Sono denominati *option1*, *option2* e *option3*. L'oggetto *option3* viene selezionato per primo.

```
          SelectFormElement list = new SelectFormElement("list1");
SelectOption option1 = list.addOption("Option1", "opt1");
SelectOption option2 = list.addOption("Option2", "opt2", false);
SelectOption option3 = new SelectOption("Option3", "opt3", true);
list.addOption(option3);        System.out.println(list.getTag());
```

L'esempio di codice precedente produce la tag HTML che segue:

```
<select name="list1">
<option value="opt1">Option1</option>
<option value="opt2">Option2</option>
<option value="opt3" selected="selected">Option3</option>
</select>
```

Classe RadioFormInputGroup:

La classe RadioFormInputGroup rappresenta un gruppo di oggetti RadioFormInput.Un utente può selezionare solo uno degli oggetti RadioFormInput da un RadioFormInputGroup.

I metodi della classe RadioFormInputGroup consentono di gestire diversi attributi di un gruppo di pallini. Con questi metodi è possibile:

- Aggiungere un pallino
- Eliminare un pallino
- Richiamare o impostare il nome del gruppo di pallini

Il seguente esempio crea un gruppo di pallini:

```
// Creare alcuni pallini.
RadioFormInput radio0 = new RadioFormInput("age", "kid", "0-12", true);
RadioFormInput radio1 = new RadioFormInput("age", "teen", "13-19", false);
RadioFormInput radio2 = new RadioFormInput("age", "twentysomething", "20-29", false);
RadioFormInput radio3 = new RadioFormInput("age", "thirtysomething", "30-39", false);
      // Creare un gruppo di pallini e aggiungerli.
RadioFormInputGroup ageGroup = new RadioFormInputGroup("age");
    ageGroup.add(radio0);
    ageGroup.add(radio1);
    ageGroup.add(radio2);
    ageGroup.add(radio3);
    System.out.println(ageGroup.getTag());
```

L'esempio di codice precedente crea il seguente codice HTML:

```
<input type="radio" name="age" value="kid" checked="checked" /> 0-12
<input type="radio" name="age" value="teen" /> 13-19
<input type="radio" name="age" value="twentysomething" /> 20-29
<input type="radio" name="age" value="thirtysomething" /> 30-39
```

Classe HTMLHead

La classe HTMLHead rappresenta una tag iniziale HTML. La sezione iniziale di una pagina HTML mostra una tag iniziale di apertura e di chiusura che solitamente contiene altre tag. Solitamente, la tag iniziale contiene una tag del titolo a possibilmente tag meta.

I programmi di creazione per HTMLHead consentono di creare una tag iniziale vuota, che contiene una tag del titolo che contiene una tag del titolo e una tag meta. E' possibile aggiungere facilmente tag meta e di titolo all'oggetto HTMLHead vuoto.

I metodi per la classe HTMLHead includono l'impostazione e il richiamo delle tag meta e di titolo della pagina. Definire il contenuto delle tag meta utilizzando la classe HTMLMeta. Per ulteriori informazioni sulla classe HTMLMeta, consultare la seguente pagina:

HTMLMeta

Il seguente codice mostra un metodo di creazione di una tag HTMLHead:

```
// Creare un HTMLHead vuoto.
HTMLHead head = new HTMLHead("My Main Page");
```

```
// Aggiungere il titolo.
head.setTitle("My main page");

// Definire le informazioni meta e aggiungerle all'HTMLHead.
HTMLMeta meta = new HTMLMeta("Content-Type", "text/html; charset=iso-8859-1");
head.addMetaInformation(meta);
```

Questa è l'emissione della tag HTMLHead di esempio:

```
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1"/>
<title>My main page</title>
</head>
```

Documentazione di riferimento Javadoc

Per ulteriori informazioni sull'utilizzo della classe HTMLHead, consultare la seguente documentazione di riferimento Javadoc:

HTMLHead

Classe HTMLHeading

La classe HTMLHeading rappresenta un'intestazione HTML. Ogni intestazione dispone di un proprio allineamento e un livello da 1 (font più grande, maggior rilievo) a 6.

I metodi per la classe HTMLHeading includono:

- Richiamare e impostare un testo dall'intestazione
- Richiamare e impostare il livello dell'intestazione
- Richiamare e impostare l'allineamento dell'intestazione
- Richiamare e impostare la direzione dell'interpretazione del testo
- Richiamare e impostare la lingua della voce immessa
- Richiamare una rappresentazione di stringa dell'oggetto HTMLHeading

Esempio: creazione degli oggetti HTMLHeading

L'esempio che segue crea tre oggetti HTMLHeading:

```
// Creare e visualizzare tre oggetti HTMLHeading.
HTMLHeading h1 = new HTMLHeading(1, "Heading", HTMLConstants.LEFT);
HTMLHeading h2 = new HTMLHeading(2, "Subheading", HTMLConstants.CENTER);
HTMLHeading h3 = new HTMLHeading(3, "Item", HTMLConstants.RIGHT);
System.out.print(h1 + "\r\n" + h2 + "\r\n" + h3);
```

L'esempio precedente produce le seguenti tag:

```
<h1 align="left">Heading</h1>
<h2 align="center">Subheading</h2>
<h3 align="right">Item</h3>
```

Classe HTMLHyperlink

La classe HTMLHyperlink rappresenta una tag hyperlink HTML.E' possibile utilizzare la classe HTMLHyperlink per creare un collegamento all'interno della pagina HTML. Con questa classe è possibile richiamare e impostare vari attributi hyperlink, incluso:

- Richiamare o impostare Uniform Resource Identifier per il collegamento
- Richiamare o impostare il titolo del collegamento
- Richiamare o impostare la frame di destinazione per il collegamento

La classe HTMLHyperlink può stampare tutto l'hyperlink con proprietà definite, in modo che sia possibile utilizzare l'emissione nella pagina HTML.

Quanto segue è un esempio relativo a HTMLHyperlink:

```
// Creare un hyperlink HTML nell'home page IBM Toolbox per Java.
HTMLHyperlink toolbox =
   new HTMLHyperlink("http://www.ibm.com/as400/toolbox", "IBM Toolbox for Java home page");
toolbox.setTarget(TARGET_BLANK);

// Visualizzare la tag del collegamento toolbox.
System.out.println(toolbox.toString());
```

Il codice precedente produce la tag che segue:

```
<a href="http://www.ibm.com/as400/toolbox">IBM Toolbox for Java home page</a>
```

Classe HTMLImage

La classe HTMLImage consente di creare tag immagine per la pagina HTML.La classe HTMLImage fornisce metodi che consentono di richiamare e impostare attributi immagine, inclusi:

- Richiamare o impostare l'altezza dell'immagine.
- Richiamare o impostare la larghezza dell'immagine.
- Richiamare o impostare il nome dell'immagine.
- Richiamare o impostare il testo alternativo dell'immagine.
- Richiamare o impostare lo spazio orizzontale intorno all'immagine.
- Richiamare o impostare lo spazio verticale intorno all'immagine.
- Richiamare o impostare i riferimenti assoluti o relativi dell'immagine.
- Richiamare una rappresentazione stringa dell'oggetto HTMLImage

L'esempio che segue mostra un metodo di creazione di un oggetto HTMLImage:

```
// Creare HTMLImage.
HTMLImage image = new HTMLImage("http://myWebSite/picture.gif", "Alternate text for this graphic");
    image.setHeight(94);
    image.setWidth(105);
    System.out.println(image);
```

L'istruzione di stampa produce la tag che segue su una riga singola. La frammentazione del testo è stata effettuata per soli scopi di visualizzazione.

```
<img src="http://myWebSite/picture.gif"
     alt="Alternate text for this graphic"
     height="94" width="105" />
```

Classi HTMLList

Le classi HTMLList consentono di creare facilmente gli elenchi all'interno delle pagine HTML. Queste classi forniscono metodi per richiamare e impostare vari attributi degli elenchi e le voci in esse contenute.

In particolare, la classe principale HTMLList fornisce un metodo per produrre una elenco compatto che visualizza le voci nel minor spazio verticale possibile.

- I metodi per HTMLList includono:
  - Comprimere l'elenco
  - Aggiungere e rimuovere voci dall'elenco
  - Aggiungere e rimuovere elenchi dall'elenco (rendendo possibile la nidificazione degli elenchi stessi)
- I metodi per HTMLListItem includono:
  - Richiamare e impostare il contenuto della voce

  - Richiamare e impostare la direzione dell'interpretazione del testo
  - Richiamare e impostare la lingua della voce immessa

Utilizzare le sottoclassi HTMLList e HTMLListItem per creare gli elenchi HTML:
- OrderedList e OrderedListItem
- UnorderedList e UnorderedListItem

Per gli snippet di codifica, consultare gli esempi che seguono:
- Esempio: creazione di elenchi ordinati
- Esempio: creazione di elenchi non ordinati
- Esempio: creazione di elenchi nidificati

OrderedList e OrderedListItem

Utilizzare le classi OrderedList e OrderedListItem per creare elenchi ordinati nelle pagine HTML.
- I metodi per OrderedList includono:
  - Richiamare e impostare il numero iniziale per la prima voce nell'elenco
  - Richiamare e impostare il tipo (o stile) per i numeri della voce
- I metodi per OrderedListItem includono:
  - Richiamare e impostare il numero per la voce
  - Richiamare e impostare il tipo (o stile) per il numero della voce

Utilizzando i metodi in OrderedListItem, è possibile sostituire la numerazione e il tipo per una voce specifica nell'elenco.

Consultare l'esempio per creare elenchi ordinati.

UnorderedList e UnorderedListItem

Utilizzare le classi UnorderedList e UnorderedListItem per creare elenchi non ordinati nelle pagine HTML.
- I metodi per UnorderedList includono:
  - Richiamare e impostare il tipo (o stile) per le voci
- I metodi per UnorderedListItem includono:
  - Richiamare e impostare il tipo (o stile) per la voce

Consultare l'esempio per creare elenchi non ordinati.

Esempi: utilizzo delle classi HTMLList

Gli esempi che seguono mostrano come utilizzare classi HTMLList per creare elenchi ordinati, non ordinati e nidificati.

Esempio: creazione di elenchi ordinati

L'esempio che segue crea un elenco ordinato:

```
          // Creare un'OrderedList.
OrderedList oList = new OrderedList(HTMLConstants.SMALL_ROMAN);
          // Creare l'OrderedListItems.
OrderedListItem listItem1 = new OrderedListItem();
OrderedListItem listItem2 = new OrderedListItem();
          // Impostare i dati in OrderedListItems.
listItem1.setItemData(new HTMLText("First item"));
```

```
listItem2.setItemData(new HTMLText("Second item"));
          // Aggiungere le voci di elenco all'OrderedList.
oList.addListItem(listItem1);
oList.addListItem(listItem2);
System.out.println(oList.getTag());
```

L'esempio precedente produce le seguenti tag:

```
<ol type="i">
<li>First item</li>
<li>Second item</li>
</ol>
```

Esempio: creazione di elenchi non ordinati

L'esempio che segue crea un elenco non ordinato:

```
          // Creare un'UnorderedList.
UnorderedList uList = new UnorderedList(HTMLConstants.SQUARE);
          // Creare l'UnorderedListItems.
UnorderedListItem listItem1 = new UnorderedListItem();
UnorderedListItem listItem2 = new UnorderedListItem();
          // Impostare i dati in UnorderedListItems.
listItem1.setItemData(new HTMLText("First item"));
listItem2.setItemData(new HTMLText("Second item"));
          // Aggiungere le voci di elenco all'UnorderedList.
uList.addListItem(listItem1);
uList.addListItem(listItem2);
System.out.println(uList.getTag());
```

L'esempio precedente produce le seguenti tag:

```
<ul type="square">
<li>First item</li>
<li>Second item</li>
</ul>
```

Esempio: creazione di elenchi nidificati

L'esempio che segue crea un elenco nidificato:

```
          // Creare un'UnorderedList.
UnorderedList uList = new UnorderedList(HTMLConstants.SQUARE);
          // Creare ed impostare i dati per UnorderedListItems.
UnorderedListItem listItem1 = new UnorderedListItem();
UnorderedListItem listItem2 = new UnorderedListItem();
listItem1.setItemData(new HTMLText("First item"));
listItem2.setItemData(new HTMLText("Second item"));
          // Aggiungere le voci di elenco all'UnorderedList.
uList.addListItem(listItem1);
uList.addListItem(listItem2);

          // Creare un'OrderedList.
OrderedList oList = new OrderedList(HTMLConstants.SMALL_ROMAN);
          // Creare l'OrderedListItems.
OrderedListItem listItem1 = new OrderedListItem();
OrderedListItem listItem2 = new OrderedListItem();
OrderedListItem listItem3 = new OrderedListItem();
          // Impostare i dati in OrderedListItems.
listItem1.setItemData(new HTMLText("First item"));
listItem2.setItemData(new HTMLText("Second item"));
listItem3.setItemData(new HTMLText("Third item"));
          // Aggiungere le voci di elenco all'OrderedList.
oList.addListItem(listItem1);
oList.addListItem(listItem2);
          // Aggiungere (nidificare) l'elenco non ordinato in OrderedListItem2
oList.addList(uList);
```

```
                // Aggiungere un'altra OrderedListItem all'OrderedList
                // dopo l'UnorderedList nidificata.
        oList.addListItem(listItem3);
        System.out.println(oList.getTag());
```

L'esempio precedente produce le seguenti tag:

```
<ol type="i">
<li>First item</li>
<li>Second item</li>
<ul type="square">
<li>First item</li>
<li>Second item</li>
</ul>
<li>Third item</li>
</ol>
```

Classe HTMLMeta

La classe HTMLMeta rappresenta informazioni Meta utilizzate in una tag HTMLHead.Gli attributi nelle tag META vengono utilizzati per identificare, indicizzare e definire informazioni nel documento HTML.

Gli attributi della tag META includono:

- NAME - il nome associato al contenuto della tag META
- CONTENT - i valori associati all'attributo NAME
- HTTP-EQUIV - le informazioni raccolte dai server HTTP per le intestazioni dei messaggi di risposta
- LANG - la lingua
- URL - utilizzato per reindirizzare l'utente dalla pagina corrente ad un'altra URL

Ad esempio, per supportare il motore di ricerca nell'individuazione del contenuto di una pagina, è possibile utilizzare la tag META che segue:

```
<META name="keywords" lang="en-us" content="games, cards, bridge">
```

E' inoltre possibile utilizzare HTMLMeta per reindirizzare l'utente da una pagina ad un'altra.

I metodi per la classe HTMLMeta includono:

- Richiamare e impostare l'attributo NAME
- Richiamare e impostare l'attributo CONTENT
- Richiamare e impostare l'attributo HTTP-EQUIV
- Richiamare e impostare l'attributo LANG
- Richiamare e impostare l'attributo URL

Esempio: creazione di tag META

L'esempio che segue crea due tag META:

```
// Creare una tag META per assistere i motori di ricerca nella determinazione del contenuto della pagina.
HTMLMeta meta1 = new HTMLMeta();
meta1.setName("keywords");
meta1.setLang("en-us");
meta1.setContent("games, cards, bridge");
// Creare una tag META utilizzata dalle cache per stabilire quando aggiornare la pagina.
HTMLMeta meta2 = new HTMLMeta("Expires", "Mon, 01 Jun 2000 12:00:00 GMT");
System.out.print(meta1 + "\r\n" + meta2);
```

L'esempio precedente produce le seguenti tag:

```
<meta name="keywords" content="games, cards, bridge">
<meta http-equiv="Expires" content="Mon, 01 Jun 2000 12:00:00 GMT">
```

Classe HTMLParameter

La classe HTMLParameter rappresenta i parametri che è possibile utilizzare con la classe HTMLServlet.Ogni parametro dispone del suo nome e valore.

I metodi per la classe HTMLParameter includono:

- Richiamare e impostare il nome del parametro
- Richiamare e impostare il valore del parametro

Esempio: creazione di tag HTMLParameter

L'esempio che segue crea una tag HTMLParameter:

```
                  // Creare un HTMLServletParameter.
     HTMLParameter parm = new HTMLParameter ("age", "21");
          System.out.println(parm);
```

Il precedente esempio produce la seguente tag:

```
     <param name="age" value="21">
```

Classe HTMLServlet

La classe HTMLServlet rappresenta il lato server.L'oggetto servlet specifica il nome del servlet e, facoltativamente, la sua posizione. E' inoltre possibile scegliere di utilizzare la posizione predefinita sul sistema locale.

La classe HTMLServlet gestisce la classe HTMLParameter, che specifica i parametri disponibili per il servlet.

I metodi per la classe HTMLServlet includono:

- Aggiungere e rimuovere HTMLParameters dalla tag servlet
- Richiamare e impostare la posizione del servlet
- Richiamare e impostare il nome del servlet
- Richiamare e impostare il testo alternativo del servlet

Esempio: creazione di tag HTMLServlet

L'esempio che segue crea una tag HTMLServlet

```
               // Creare un HTMLServlet.
          HTMLServlet servlet = new HTMLServlet("myServlet", "http://server:port/dir");

               // Creare un parametro, quindi aggiungerlo al servlet.
          HTMLParameter param = new HTMLParameter("parm1", "value1");
          servlet.addParameter(param);

               // Creare ed aggiungere un secondo parametro
          HTMLParameter param2 = servlet.add("parm2", "value2");

               // Creare il testo alternativo se il server Web non supporta la tag del servlet.
     servlet.setText("The Web server providing this page does not support the SERVLET tag.")
;
     System.out.println(servlet);
```

L'esempio precedente produce le seguenti tag:

```
<servlet name="myServlet" codebase="http://server:port/dir">
<param name="parm1" value="value1">
<param name="parm2" value="value2">
     The Web server providing this page does not support the SERVLET tag.
</servlet>
```

Classi HTMLTable

La classe HTMLTable consente di impostare facilmente tabelle che è possibile utilizzare nelle pagine HTML.Questa classe fornisce metodi per richiamare e impostare vari attributi della tabella, incluso:

- Richiamare e impostare la larghezza del bordo
- Richiamare il numero di righe nella tabella
- Aggiungere una colonna o unariga alla fine della tabella
- Rimuovere una colonna o una riga da una colonna specifica o dalla riga stessa

La classe HTMLTable utilizza altre classi HTML per semplificare la creazione della tabella. Le altre classi HTML che aiutano a creare tabelle sono:

- HTMLTableCell: crea una cella nella tabella
- HTMLTableRow: crea una riga nella tabella
- HTMLTableHeader: crea una cella di intestazione nella tabella
- HTMLTableCaption: crea un titolo per la tabella

Esempio: utilizzo delle classi HTMLTable

Il seguente esempio mostra come utilizzare le classi HTMLTable:

"Esempio: utilizzo delle classi HTMLTable" a pagina 613

Classe HTMLTableCell:

La classe HTMLTableCell prende qualsiasi oggettoHTMLTagElement come immissione e crea la tag cella della tabella con l'elemento specificato.L'elemento può essere impostato sul programma di creazione o attraverso due metodi setElement().

Molti attributi cella possono essere richiamati o aggiornati utilizzando metodi forniti nella classe HTMLTableCell. Alcune delle azioni che è possibile eseguire con questi metodi sono:

- Richiamare o impostare la dimensione riga
- Richiamare o impostare l'altezza della cella
- Stabilire se i dati della cella utilizzeranno le normali convenzioni di interruzione riga HTML

L'esempio che segue crea un oggetto HTMLTableCell e visualizza la tag:

```
//Creare un oggetto HTMLHyperlink.
HTMLHyperlink link = new HTMLHyperlink("http://www.ibm.com",
                                       "IBM Home Page");
HTMLTableCell cell = new HTMLTableCell(link);
cell.setHorizontalAlignment(HTMLConstants.CENTER);
System.out.println(cell.getTag());
```

Il metodo precedente getTag() fornisce l'emissione dell'esempio:

<td align="center">IBM Home Page</td>

Classe HTMLTableRow:

La classe HTMLTableRow crea una riga nella tabella.Questa classe fornisce vari metodi per richiamare e impostare attributi di riga. Alcune azioni che si possono eseguire con questi metodi sono:

- Aggiungere o rimuovere una colonna dalla riga
- Richiamare i dati della colonna sull'indice della colonna specificato
- Richiamare l'indice della colonna per la colonna con la cella specificata.
- Richiamare il numero delle colonne in una riga
- Impostare allineamenti orizzontali e verticali

Quanto segue è un esempio per HTMLTableRow:

```
// Creare una riga e impostare l'allineamento.
HTMLTableRow row = new HTMLTableRow();
   row.setHorizontalAlignment(HTMLTableRow.CENTER);

// Creare e aggiungere le informazioni di colonna alla riga.
HTMLText account = new HTMLText(customers_[rowIndex].getAccount());
HTMLText name = new HTMLText(customers_[rowIndex].getName());
HTMLText balance = new HTMLText(customers_[rowIndex].getBalance());

   row.addColumn(new HTMLTableCell(account));
row.addColumn(new HTMLTableCell(name));
row.addColumn(new HTMLTableCell(balance));

// Aggiungere la riga ad un oggetto HTMLTable (presupporre che la tabella esiste già).
table.addRow(row);
```

Classe HTMLTableHeader:

La classe HTMLTableHeader eredita dalla classe HTMLTableCell.Crea un tipo di cella specifico, la cella di ntestazione, fornendo una cella **<th>** invece di una cella **<td>**. Così come la classe HTMLTableCell, è possibile richiamare vari metodi per aggiornare o richiamare gli attributi della cella di intestazione.

Quanto segue è un esempio per HTMLTableHeader:

```
// Creare le intestazioni tabella.
HTMLTableHeader account_header = new HTMLTableHeader(new HTMLText("ACCOUNT"));
HTMLTableHeader name_header = new HTMLTableHeader(new HTMLText("NAME"));
HTMLTableHeader balance_header = new HTMLTableHeader();
HTMLText balance = new HTMLText("BALANCE");
balance_header.setElement(balance);

// Aggiungere le intestazioni tabella ad un oggetto HTMLTable (presupporre che la tabella esista già).
table.addColumnHeader(account_header);
table.addColumnHeader(name_header);
table.addColumnHeader(balance_header);
```

Classe HTMLTableCaption:

La classe HTMLTableCaption crea un titolo per la tabella HTML.La classe fornisce metodi per aggiornare e richiamare gli attributi del titolo. Ad esempio è possibile utilizzare il metodo setAlignment() per specificare su quale parte della tabella allineare il titolo. Quanto segue è un esempio per HTMLTableCaption:

```
// Creare un oggetto HTMLTableCaption predefinito ed impostare il testo del titolo.
      HTMLTableCaption caption = new HTMLTableCaption();
caption.setElement("Customer Account Balances - January 1, 2000");

// Aggiungere un titolo tabella ad un oggetto HTMLTable (presupporre che la tabella esista già).
      table.setCaption(caption);
```

Classe HTML Text

La classe HTMLText consente di accedere alle proprietà del testo per la pagina HTML.Utilizzando la classe HTMLText, è possibile richiamare, impostare e controllare lo stato di molti attributi testo, incluso:

- Richiamare o impostare la dimensione del font
- Impostare l'attributo bold su on (true) o off (false) o determinare se è già impostato su on
- Impostare l'attributo underscore su on (true) o off (false) o determinare se è già impostato su on
- Richiamare o impostare l'allineamento orizzontale del testo

L'esempio che segue mostra come creare un oggetto HTMLText e impostare l'attributo bold su on e la dimensione del font su 5.

```
HTMLText text = new HTMLText("IBM");
     text.setBold(true);
text.setSize(5);
System.out.println(text.getTag());
```

La specifica di stampa produce la tag che segue:

```
<font size="5"><b>IBM</b></font>
```

Quando si utilizza questa tag in una pagina HTML, essa viene visualizzata in questo modo:

IBM

Classi HTMLTree

Le classi HTMLTree consentono di impostare facilmente una gerarchia ad albero di elementi HTML che è possibile utilizzare nelle pagine HTML.Questa classe fornisce metodi per richiamare e impostare vari attributi dell'albero, in aggiunta a metodi che consentono di:

- Richiamare e impostare la richiesta servlet HTTP
- Aggiungere un HTMLTreeElement o FileTreeElement all'albero
- Rimuovere un HTMLTreeElement o FileTreeElement dall'albero

Le classi HTMLTree utilizzano altre classi HTML che facilitano la creazione della gerarchia ad albero:

- HTMLTreeElement: crea un elemento ad albero
- FileTreeElement: crea un elemento ad albero del file
- FileListElement: crea un elemento elenco file
- FileListRenderer: restituisce l'elenco di file e indirizzari

Esempi: utilizzo delle classi HTMLTree

Gli esempi che seguono mostrano differenti metodi di utilizzo delle classi HTMLTree.

- "Esempio: utilizzo delle classi HTMLTree" a pagina 603
- "Esempio: creazione di una gerarchia ad albero IFS"

Esempio: creazione di una gerarchia ad albero IFS:

L'esempio che segue è costituito da tre file che, insieme, mostrano come è possibile creare una gerarchia ad albero IFS. L'esempio utilizza delle frame per visualizzare HTMLTree e FileListElement in un servlet.

- FileTreeExample.java - crea le frame HTML e avvia il servlet
- TreeNav.java - crea e gestisce l'albero
- TreeList.java - visualizza il contenuto delle selezioni effettuate nella classe TreeNav.java

Classe HTMLTreeElement:

La classe HTMLTreeElement rappresenta un elemento gerarchico all'interno di HTMLTree o altri HTMLTreeElements.

Molti attributi dell'elemento ad albero possono essere richiamati o aggiornati utilizzando metodi forniti nella classe HTMLTreeElement. Alcune delle azioni che è possibile eseguire con questi metodi sono:

- Richiamare o impostare il testo visibile dell'elemento ad albero
- Richiamare o impostare l'URL per l'icona espansa e compressa
- Stabilire se l'elemento albero verrà espanso

L'esempio che segue crea un oggetto HTMLTreeElement e visualizza la tag:

```
// Creare un HTMLTree.
HTMLTree tree = new HTMLTree();

// Creare un HTMLTreeElement principale.
HTMLTreeElement parentElement = new HTMLTreeElement();
parentElement.setTextUrl(new HTMLHyperlink("http://myWebPage", "My Web Page"));


// Creare un HTMLTreeElement secondario.
HTMLTreeElement childElement = new HTMLTreeElement();
childElement.setTextUrl(new HTMLHyperlink("http://anotherWebPage", "Another Web Page"));
parentElement.addElement(childElement);

// Aggiungere un elemento all'albero.
tree.addElement(parentElement);
System.out.println(tree.getTag());
```

Il metodo getTag() nell'esempio precedente crea tag HTML come quelle che seguono:

```
<table cellpadding="0" cellspacing="3">
<tr>
<td><font color="#0000FF"><u>-</u></font> </td>
<td><font color="#0000FF"><u>Pagina Web utente</u></font></td>
</tr>

<tr>
<td> </td>
<td>
<table cellpadding="0" cellspacing="3">
<tr>
<td><font color="#0000FF"><u>-</u></font> </td>
<td><font color="#0000FF"><u>Altra pagina Web</u></font> </td>
</tr>
</table></td>
</tr>
</table>
```

Classe FileTreeElement:

La classe FileTreeElement rappresenta l'IFS in una vista HTMLTree.

Molti attributi dell'elemento ad albero possono essere richiamati o aggiornati utilizzando metodi forniti nella classe HTMLTreeElement. L'utente può inoltre richiamare e impostare il nome e il percorso delle unità condivise NetServer.

Alcune azioni che questi metodi consentono di eseguire sono:

- Richiamare o impostare l'URL per l'icona espansa e compressa (metodo ereditato)
- Stabilire se l'elemento ad albero verrà espanso (metodo ereditato)
- Richiamare o impostare il nome dell'unità condivisa NetServer
- Richiamare o impostare il percorso dell'unità condivisa NetServer

Esempio: utilizzo di FileTreeElement

L'esempio che segue crea un oggetto FileTreeElement e visualizza la tag:

```
             // Creare un HTMLTree.
             HTMLTree tree = new HTMLTree();

               // Creare un oggetto URLParser.
            URLParser urlParser = new URLParser(httpServletRequest.getRequestURI());

            // Creare un oggetto AS400.
             AS400 system = new AS400(mySystem, myUserId, myPassword);

             // Creare un oggetto IFSJavaFile.
             IFSJavaFile root = new IFSJavaFile(system, "/QIBM");

             // Creare un oggetto DirFilter e richiamare gli indirizzari.
                DirFilter filter = new DirFilter();
             File[] dirList = root.listFiles(filter);

             for (int i=0; i < dirList.length; i++)
      {

                     // Creare un FileTreeElement.
                      FileTreeElement node = new FileTreeElement(dirList[i]);

                     // Impostare l'URL icona.
                      ServletHyperlink sl = new ServletHyperlink(urlParser.getURI());
                      sl.setHttpServletResponse(resp);
                     element.setIconUrl(sl);

                     // Aggiungere un FileTreeElement all'albero.
                     tree.addElement(element);
      }

             System.out.println(tree.getTag());
```

Il metodo precedente getTag() fornisce l'emissione dell'esempio.

Classe FileListElement:

La classe FileListElement consente di creare un elemento dell'elenco di file che rappresenta il contenuto di un indirizzario IFS (integrated file system).

E' possibile utilizzare l'oggetto FileListElement per rappresentare il contenuto di una unità condivisa NetServer, richiamando e impostando il nome e il percorso delle unità condivise NetServer.

La classe FileListElement fornisce metodi che consentono di:

- Elencare e ordinare gli elementi dell'elenco di file
- Richiamare e impostare la richiesta servlet HTTP
- Richiamare e impostare FileListRenderer
- Richiamare e impostare HTMLTable con cui visualizzare l'elenco di file
- Richiamare o impostare il nome di un'unità condivisa NetServer
- Richiamare o impostare il percorso di un'unità condivisa NetServer

E' possibile utilizzare la classe FileListElement con altre classi nel pacchetto html:

- Con un FileListRenderer, è possibile specificare il tipo di visualizzazione desiderata dell'elenco di file
- Con la classe FileTreeElement, è possibile creare un elenco gerarchico di file IFS o di file condivisi NetServer

Il javadoc FileListElement mostra come creare e visualizzare un oggetto FileListElement.

Esempio: utilizzo di FileListElement per creare una gerarchia ad albero IFS

L'esempio seguente mostra come utilizzare la classe FileListElement con classi HTMLTree (FileTreeElement e HTMLTreeElement) per creare una gerarchia ad albero IFS. L'esempio include inoltre il codice per impostare il percorso dell'unità condivisa NetServe.

"Esempio: creazione di una gerarchia ad albero IFS" a pagina 217

Classe FileListRenderer:

La classe FileListRenderer trasforma qualsiasi campo per gli oggetti file (indirizzari e file) in un FileListElement.

La classe FileListRenderer offre metodi che consentono di eseguire le azioni che seguono:

- Richiamare il nome dell'indirizzario.
- Richiamare il nome del file
- Richiamare il nome dell'indirizzario principale
- Restituire i dati della riga che si desidera visualizzare nel FileListElement

Questo esempio crea un oggetto FileListElement con un renderer:

```
// Creare un FileListElement.
FileListElement fileList = new FileListElement(sys, httpservletrequest);

// Impostare il renderer specifico su questo servlet, che estende
// FileListRenderer e sostituisce i metodi applicabili.
fileList.setRenderer(new myFileListRenderer(request));
```

Se non si desidera utilizzare il renderer predefinito, è possibile estendere FileListRenderer e sostituire metodi o crearne di nuovi. Ad esempio, potrebbe essere necessario assicurarsi di impedire l'inoltro di nomi di specifici indirizzari o file con alcune estensioni al FileListElement. Estendendo la classe e sostituendo il metodo appropriato, è possibile restituire null per questi file e indirizzari, assicurandosi che non vengano visualizzati.

Per personalizzare interamente le righe in un FileListElement, utilizzare il metodo getRowData(). Un esempio di personalizzazione di dati della riga utilizzando getRowData() potrebbe essere quello di aggiungere una colonna ai dati riga o di disporre diversamente le colonne. Quando la funzionalità predefinita di FileListRenderer è soddisfacente, non è necessaria una programmazione aggiuntiva perché la classe FileListElement crea un FileListRenderer predefinito.

Classi ReportWriter

Il pacchetto com.ibm.as400.util.reportwriter fornisce classi che consentono di utilizzare iSeries per accedere più facilmente e per formattare i dati da un file sorgente XML o i dati prodotti dai servlet o dalle JSP (JavaServer Pages)(TM). Il pacchetto reportwriter offre un metodo appropriato per denominare tre pacchetti differenti ma correlati:

- com.ibm.as400.util.reportwriter.pclwriter
- com.ibm.as400.util.reportwriter.pdfwriter
- com.ibm.as400.util.reportwriter.processor

Questi pacchetti includono una varietà di classi che consentono di formattare flussi di dati XML e creare prospetti nei suddetti formati. Accertarsi di disporre dei file jar necessari in CLASSPATH, inclusi un programma di analisi XML ed un processore XSLT. Per ulteriori informazioni, consultare le seguenti pagine:

File jar

"Programma di analisi XML e processore XSLT" a pagina 417

.

Le classi context (nei pacchetti pclwriter e pdfwriter) definiscono i metodi necessari alle classi ReportProcessor per rendere i dati XML e JSP nel formato scelto:

- Utilizzare PCLContext in combinazione con una classe ReportWriter per creare un prospetto nel formato PCL (Printer Control Language di Hewlett Packard).
- Utilizzare PDFContext in combinazione con una classe ReportWriter per creare un prospetto nel formato PDF (Portable Document Format di Adobe).

Le classi ReportProcessor (nel pacchetto del processore) consentono di creare prospetti formattati dalle informazioni che l'applicazione raccoglie dai dati sorgente XML, dai servlet Java e da JSP (JavaServer Pages).

- Utilizzare la classe JSPReportProcessor per richiamare i dati dai servlet e dalle pagine JSP per produrre i prospetti nei formati disponibili (contesti).
- Utilizzare la classe XSLReportProcessor per elaborare i dati XML con i fogli di stile XSL per produrre i prospetti nei formati disponibili (contesti).

Classi Context

Le classi Context supportano specifici formati di dati che, in combinazione con le classi OutputQueue e SpooledFileOutputStream, consentono alle classi ReportWriter di creare prospetti in quel formato e inseriscono tali prospetti in un file di spool.

L'applicazione deve solo creare un'istanza della classe Context, che le classi ReportWriter poi utilizzano per creare i prospetti. L'applicazione non richiama mai direttamente i metodi in ogni Classe Context. Il PCLContext e i metodi PDFContext devono essere utilizzati internamente dalle classi ReportWriter.

La costruzione di una istanza della Classe di contesto richiede un OutputStream (dal pacchetto java.io) e un PageFormat (dal pacchetto java.awt.print). I seguenti esempi mostrano come si può costruire ed utilizzare le classi Context con altre classi ReportWriter per creare prospetti:

Esempio: utilizzo di XSLReportProcessor con PCLContext

Esempio: utilizzo di JSPReportProcessor con PDFContext

Classe JSPReportProcessor

La classe JSPReportProcessor consente di creare un documento o un prospetto dal contenuto di JavaServer Page(TM) (JSP) o Java servlet.

Utilizzare questa classe per ottenere una JSP o un servlet da una determinata URL e creare un documento dal contenuto. E' necessario che la JSP o il servlet forniscano i dati del documento, inclusi gli oggetti di formattazione XSL. E' necessario specificare il contesto di emissione e il sorgente dei dati di immissione JSP prima di poter creare una qualsiasi pagina del documento. E' possibile, poi, convertire i dati del prospetto in un determinato formato di flusso di dati di emissione.

La classe JSPReportProcessor consente di:

- Elaborare il prospetto
- Impostare una URL come mascherina

I seguenti esempi mostrano come è possibile utilizzare le classi JSPReportProcessor e PDFContext per creare un prospetto. Gli esempi includono sia il codice Java che JSP, che è possibile visualizzare utilizzando i seguenti collegamenti. E' inoltre possibile scaricare un file ZIP che contiene i file sorgente di esempio JSP, XML e XSL per entrambi gli esempi JSPReportProcessor e XSLReportProcessor:

- "Esempio: utilizzo di JSPReportProcessor con PDFContext" a pagina 624
- "Esempio: file JSP di esempio JSPReportProcessor" a pagina 626

Per ulteriori informazioni su JSP, consultare il sito web Tecnologia Java Server Pages  .

Classe XSLReportProcessor

La classe XSLReportProcessor consente all'utente di creare un documento o un prospetto trasformando e formattando i dati sorgenti XML utilizzando un foglio di stile XSL.Utilizzare questa classe per creare il prospetto utilizzando un foglio di stile XSL contenente FO (formatting objects) conformi alla specifica XSL. In seguito, utilizzare una classe Context per convertire i dati del prospetto in un formato specifico del flusso dati di emissione.

La classe XSLReportProcessor consente di:

- Impostare il foglio di stile XSL
- Impostare Sorgente dati XML
- Impostare Sorgente FO XSL
- Elaborare un prospetto

Esempi

I seguenti esempi mostrano come è possibile utilizzare le classi XSLReportProcessor e PCLContext per creare un prospetto. Gli esempi includono il codice Java, XML e XSL, che è possibile visualizzare utilizzando i seguenti collegamenti. E' inoltre possibile scaricare un file zip che contiene i file sorgente XML, XSL e JSP di esempio per gli esempi sia XSLReportProcessor che JSPReportProcessor:

- Esempio: utilizzo di XSLReportProcessor con PCLContext
- Esempio: file XML di esempio XSLReportProcessor
- Esempio: file XML di esempio XSLReportProcessor

Per ulteriori informazioni su XML e XSL, consultare l'argomento XML nell'Information Center.

Classi Resource

Il pacchetto com.ibm.as400.resource fornisce una framework generica per gestire i vari oggetti e gli elenchi AS400. Tale framework fornisce un'interfaccia di programmazione coerente rispetto a tali oggetti ed elenchi.

Il pacchetto resource include le seguenti classi:

- Resource - un oggetto che rappresenta una risorsa iSeries, ad esempio un utente, una stampante, un lavoro, un messaggio o un file.Le sottoclassi concrete di Resource includono:
  - RIFSFile
  - RJavaProgram
  - RJob
  - RPrinter
  - RQueuedMessage
  - RSoftwareResource
  - RUser

Nota: Le classi NetServer nel pacchetto access sono anche sottoclassi concrete di Resource.

- ResourceList - un oggetto che rappresenta un elenco di risorse iSeries, ad esempio un elenco di utenti, stampanti, lavori, messaggi o file. Le sottoclassi concrete di Resource includono:
  - RIFSFileList
  - RJobList
  - RJobLog
  - RMessageQueue
  - RPrinterList
  - RUserList
- Presentation - un oggetto che consente di presentare informazioni su oggetti risorsa, elenchi di risorse, attributi, selezioni e ordinamenti per gli utenti finali.

Classi Resource e ChangeableResource

Le classi astratte com.ibm.as400.resource.Resource e com.ibm.as400.resource.ChangeableResource rappresentano una risorsa iSeries.

Resource

Resource è una classe astratta che fornisce un accesso generico agli attributi di ogni risorsa. Ogni attributo viene identificato utilizzando un ID attributo e ogni sottoclasse data di Resource documenterà normalmente gli ID attributo che supporta.

Resource fornisce solo un accesso di lettura ai valori dell'attributo.

IBM Toolbox per Java fornisce i seguenti oggetti risorsa:

- RIFSFile - rappresenta un file o un indirizzario nell'IFS di iSeries
- RJavaProgram - rappresenta un programma Java sull'iSeries
- RJob - rappresenta un lavoro iSeries
- RPrinter - rappresenta una stampante iSeries
- RQueuedMessage - rappresenta un messaggio in una coda di messaggi iSeries o in una registrazione lavori.
- RSoftwareResource - rappresenta un programma su licenza in iSeries
- RUser - rappresenta un utente iSeries

ChangeableResource

La classe astratta ChangeableResource, una sottoclasse di Resource, aggiunge la possibilità di modificare i valori dell'attributo di una risorsa iSeries. Le modifiche dell'attributo sono memorizzate internamente in una cache fino a quando vengono convalidate o cancellate. Ciò consente di modificare molti valori dell'attributo alla volta.

Nota: le classi NetServer nel pacchetto access sono anche sottoclassi concrete di Resource e ChangeableResource.

Esempi

I seguenti esempi illustrano come è possibile utilizzare direttamente le sottoclassi concrete di Resource e ChangeableResource e come un codice generico può interagire con qualsiasi sottoclasse Resource o ChangeableResource.

- Richiamare un valore di attributo da RUUser, una sottoclasse concreta di Resource
- Impostare i valori di attributo per RJob, una sottoclasse concreta di ChangeableResource

- Utilizzare un codice generico per accedere alle risorse

Esonero di responsabilità per gli esempi di codice

Il seguente esonero di responsabilità si applica a tutti gli esempi IBM Toolbox per Java:

> IBM fornisce una licenza non esclusiva per utilizzare ciò come esempio da cui creare funzioni simili personalizzate, in base a richieste specifiche.
>
> Tutti gli esempi di codice forniti dall'IBM hanno la sola funzione illustrativa. Questi esempi non sono stati interamente testati in tutte le condizioni. L'IBM, perciò, non intende implicita alcuna garanzia di affidabilità, manutenibilità o funzionalità di questi programmi.
>
> Tutti i programmi qui contenuti sono forniti ″COSI' COME SONO″ senza garanzie di alcun tipo. Sono espressamente smentite tutte le garanzie implicite di non violazione, di commerciabilità e idoneità per scopi specifici.

Elenchi di risorse

La classe com.ibm.as400.resource.ResourceList rappresenta un elenco di risorse iSeries. E' una classe astratta che fornisce un accesso generico al contenuto dell'elenco.

IBM Toolbox per Java fornisce i seguenti elenchi di risorse:
- RIFSFileList - rappresenta un elenco di file e indirizzari nell'IFS di iSeries
- RJobList - rappresenta un elenco di lavori iSeries
- RJobLog - rappresenta un elenco di messaggi in una registrazione lavori iSeries
- RMessageQueue - rappresenta un elenco di messaggi nella coda messaggi iSeries
- RPrinterList - rappresenta un elenco di stampanti iSeries
- RUserList - rappresenta un elenco di utenti iSeries

Un elenco di risorse è sempre o chiuso o aperto. E' necessario che l'elenco di risorse sia aperto per accedere al contenuto. Per consentire l'accesso immediato al contenuto dell'elenco e la gestione della memoria in modo efficiente, gran parte degli elenchi di risorse vengono caricati in modo incrementale.

Gli elenchi di risorse consentono di:
- Aprire l'elenco
- Chiudere l'elenco
- Accedere ad una risorsa specifica dall'elenco
- Attendere il caricamento di una particolare risorsa
- Attendere il caricamento di un elenco di risorse completo

E' anche possibile filtrare gli elenchi di risorse utilizzando i valori di selezione. Ogni valore di selezione viene identificato utilizzando un ID selezione. Allo stesso modo, è possibile ordinare gli elenchi di risorse utilizzando i valori di ordinamento. Ogni valore di ordinamento viene identificato utilizzando un ID ordinamento. Ogni sottoclasse data di ResourceList documenterà normalmente gli ID selezione e gli ID ordinamento che supporta.

Esempi

I seguenti esempi mostrano le varie modalità di gestione degli elenchi di risorse:
- Esempio: richiamo e stampa del contenuto di una ResourceList
- Esempio: utilizzo di un codice generico per accedere ad una ResourceList
- Esempio: presentazione di un elenco di risorse in un servlet (tabella HTML)

Esonero di responsabilità per gli esempi di codice

Il seguente esonero di responsabilità si applica a tutti gli esempi IBM Toolbox per Java:

> L'IBM fornisce una licenza non esclusiva per utilizzare tutti gli esempi del codice di programmazione da cui creare funzioni simili personalizzate, in base a richieste specifiche.
>
> Questo codice di esempio è fornito dall'IBM con la sola funzione illustrativa. Questi esempi non sono stati interamente testati in tutte le condizioni. IBM, perciò, non fornisce nessun tipo di garanzia o affidabilità implicita, rispetto alla funzionalità o alle funzioni di questi programmi.
>
> Tutti i programmi qui contenuti vengono forniti all'utente ″COSI' COME SONO″ senza garanzie di alcun tipo. Le garanzie implicite di non contraffazione, commerciabilità e adeguatezza a scopi specifici sono espressamente vietate.

Classe Presentation

Ad ogni oggetto risorsa, elenco di risorse e oggetto metadati è associato un oggetto com.ibm.as400.resource.Presentation che fornisce informazioni convertite, come il nome, il nome completo e l'icona.

Esempio: stampa di un elenco di risorse e dei relativi valori di ordinamento utilizzando le rispettive Presentation

E' possibile utilizzare le informazioni Presentation per presentare gli oggetti risorsa, gli elenchi di risorse, le selezioni e gli ordinamenti per gli utenti finali nel formato testo.

```
   void printCurrentSort(ResourceList resourceList) throws ResourceException
{
            // Richiamare la presentazione per la ResourceList e stamparne il nome completo.
            Presentation resourceListPresentation = resourceList.getPresentation();
            System.out.println(resourceListPresentation.getFullName());

            // Richiamare il valore di ordinamento corrente.
            Object[] sortIDs = resourceList.getSortValue();

            // Stampare ogni ID ordinamento.
            for(int i = 0; i < sortIDs.length; ++i)
    {
                    ResourceMetaData sortMetaData = resourceList.getSortMetaData(sortIDs[i]);
                    System.out.println("Sorting by " + sortMetaData.getName());
    }
}
```

Classi Security

E' possibile utilizzare le classi di sicurezza IBM Toolbox per Java per fornire collegamenti sicuri al server, verificare l'identità dell'utente e associare un utente ad un sottoprocesso di sistema operativo quando viene eseguito su un server locale. I servizi di sicurezza inclusi sono:

- JSSE (Java Secure Socket Extension) garantisce collegamenti protetti sia codificando i dati scambiati tra un client e una sessione server che eseguendo l'autenticazione server.

  Nota: informazioni sull'utilizzo di SSL (Secure Sockets Layer) sono incluse solo per compatibilità con le versioni precedenti.
- I servizi di autenticazione forniscono la capacità di:
  - Autenticare l'identità e la parola d'ordine dell'utente rispetto al registro utenti OS/400.
  - Assegnare un'identità al sottoprocesso OS/400 corrente.

SSL (Secure Sockets Layer)

SSL (Secure Sockets Layer) fornisce collegamenti sicuri tramite:

- La codifica dei dati scambiati tra una sessione client e server
- L'esecuzione dell'autenticazione del server

Nota: si consideri l'utilizzo di JSSE (Java Secure Socket Extension) invece dei seguenti metodi per fornire collegamenti protetti. Le seguenti informazioni sull'utilizzo di SSL vengono incluse solo per compatibilità con le versioni precedenti.

L'utilizzo di SSL influenza negativamente le prestazioni poiché i collegamenti SSL funzionano più lentamente dei collegamenti che non hanno codifica. Utilizzare collegamenti SSL quando la riservatezza dei dati trasferiti ha una priorità maggiore rispetto alle prestazioni, ad esempio, quando si trasferiscono dati relativi alla carta di credito o informazioni sul rendiconto bancario.

Prima di iniziare ad utilizzare SSL con IBM Toolbox per Java, è necessario considerare le responsabilità legali.

Algoritmi SSL

IBM Toolbox per Java non contiene gli algoritmi necessari per codificare e decodificare i dati. Nella V5R3, questi algoritmi vengono inviati nel programma su licenza iSeries Client Encryption (128-bit), 5722-CE3.

Nota: IBM Toolbox per Java è compatibile anche con il programma su licenza iSeries Client Encryption (56-bit), 5722-CE2, che non viene più aggiornato e non è disponibile per V5R3. Dal momento che Client Encryption (56 bit) contiene algoritmi meno potenti di Client Encryption (128 bit), è necessario considerare un aggiornamento ad una codifica a 128 bit.

Contattare il rappresentante IBM per ulteriori informazioni o per ordinare Client Encryption (128-bit), 5722-CE3.

Impostazione dell'ambiente SSL

IBM Toolbox per Java fornisce due ambienti per l'utilizzo di SSL per codificare i dati, che è necessario impostare correttamente.

- Utilizzo della codifica tra le classi IBM Toolbox per Java e i server OS/400
- Utilizzo della codifica tra il client proxy e il server proxy

Compatibilità con le precedenti versioni di IBM Toolbox per Java

La versione V5R3 di IBM Toolbox per Java, gli algoritmi di codifica e i file di classe keyring richiedono l'utilizzo dei programmi su licenza Client Encryption V5R1, V5R2 o V5R3.

Nota: se si esegue un aggiornamento da OS/400 versione V4R5 o precedenti, è necessario aggiornare il file KeyRing.class.

Quando si utilizza V5R3 IBM Toolbox per Java ed una versione compatibile di Client Encryption sul client, è possibile collegarsi a V5R1 e a versioni più recenti di OS/400. Per ulteriori informazioni sulle versioni compatibili di Client Encryption, consultare Algoritmi SSL.

Responsabilità legali SSL: Il prodotto su licenza IBM iSeries Client Encryption (128-bit) fornisce il supporto di codifica SSL Versione 3.0 utilizzando gli algoritmi di codifica a 128 bit.

Questo programma è fornito di una tecnologia di codifica di dati soggetta a speciali requisiti di licenza di esportazione del Dipartimento del commercio degli Stati Uniti. Anche altri Paesi o regioni possono disporre di requisiti di licenza di importazione ed esportazione.

Con la presente l'utente viene avvisato che l'utilizzo, o il trasferimento, da parte di utenti in un qualsiasi paese/regione dello stesso programma potrebbe essere vietato o soggetto a:

- Leggi, regole o normative speciali del governo nazionale dell'utente.
- Leggi, regole o normative speciali di esportazione del governo nazionale.

L'utente si assume tutte le responsabilità per assicurare che il programma venga utilizzato o trasferito in accordo con tutte le leggi, regole o normative applicabili di importazione ed esportazione prima e dopo la data di scadenza della licenza.

Tutti gli utenti devono ottemperare con le leggi di importazione ed esportazione degli altri paesi o delle altre regioni.

Utilizzo di SSL per codificare dati tra IBM Toolbox per Java e i server OS/400:

E' possibile utilizzare SSL per codificare dati scambiati tra le classi IBM Toolbox per Java e i server OS/400. Sul lato client, si utilizzano i file che vengono forniti dal programma su licenza IBM iSeries Client Encryption (5722-CE2 o 5722-CE3) per codificare i dati. Sul lato server, è necessario utilizzare Digital Certificate Manager OS/400 per configurare i server OS/400 per scambiare dati codificati.

Impostazione del client e del server per utilizzare SSL

Per codificare i dati che si trovano nelle classi IBM Toolbox per Java e nei server OS/400, completare le attività che seguono:

1. Impostare i server per scambiare i dati codificati.
2. Impostare il client (le classi IBM Toolbox per Java) per scambiare dati codificati. La procedura relativa a questo passo dipende dal tipo di certificato utilizzato nell'impostazione di SSL sul server:
   - Utilizzare un certificato server da un'autorità garantita
   - Utilizzare un'autocertificazione

   Nota: impostare il client utilizzando un certificato da un'autorità garantita è notevolmente più facile e veloce che utilizzare un'autocertificazione.
3. Utilizzare l'oggetto SecureAS400 per forzare IBM Toolbox per Java a codificare i dati.

   Nota: il completamento dei primi due passi riportati sopra crea solo un percorso sicuro tra il client ed il server. L'applicazione deve utilizzare l'oggetto SecureAS400 per indicare a IBM Toolbox per Java quali dati codificare. I dati che si trovano nell'oggetto SecureAS400 sono i soli dati da codificare. Se si utilizza un oggetto AS400, i dati non vengono codificati e viene utilizzato il percorso normale al server.

Impostazione dei server iSeries per utilizzare SSL:

Per impostare i server iSeries per utilizzare SSL con IBM Toolbox per Java, completare le fasi che seguono:

1. Installare quanto segue sui server iSeries:
   - IBM Cryptographic Access Provider 128-bit per iSeries, 5722-AC3, il quale fornisce la codifica lato server.
   - iSeries Client Encryption (128-bit), 5722-CE3, che fornisce le classi Java e i programmi di utilità utilizzati dalle classi IBM Toolbox per Java sul lato client.

Nota: IBM Toolbox per Java è compatibile anche con la versione V5R1 di Cryptographic Access Provider 56-bit per iSeries, 5722-AC2 e con la versione V5R1 di Client Encryption (56-bit), 5722-CE2.

2. Sostituire l'autorizzazione dell'indirizzario che contiene i file di codifica del client.
3. Richiamare e configurare il certificato server.
4. Applicare il certificato ai server iSeries riportati di seguito che vengono utilizzati da IBM Toolbox for Java:
   - QIBM_OS400_QZBS_SVR_CENTRAL
   - QIBM_OS400_QZBS_SVR_DATABASE
   - QIBM_OS400_QZBS_SVR_DTAQ
   - QIBM_OS400_QZBS_SVR_NETPRT
   - QIBM_OS400_QZBS_SVR_RMTCMD
   - QIBM_OS400_QZBS_SVR_SIGNON
   - QIBM_OS400_QZBS_SVR_FILE
   - QIBM_OS400_QRW_SVR_DDM_DRDA

Modifica dell'autorizzazione dell'indirizzario che contiene i file di codifica del client

Per aiutare l'utente a soddisfare le responsabilità legali SSL necessarie quando si utilizzano algoritmi crittografici, l'indirizzario che contiene i file viene fornito con l'autorizzazione pubblica *EXCLUDE.E' necessario modificare l'autorizzazione dell'indirizzario per consentire l'accesso solo agli utenti autorizzati a utilizzare gli algoritmi di codifica.

Utilizzare l'oggetto di sicurezza OS/400 per controllare l'accesso ai file di codifica del client completando le fasi che seguono:

1. Sul server, immettere il comando che segue:
   ```
   wrklnk '/QIBM/ProdData/HTTP/Public/jt400/*'
   ```
2. Selezionare l'opzione 9 nell'indirizzario SSL56 o SSL128.
3. Assicurarsi che *PUBLIC abbia l'autorizzazione *EXCLUDE.
4. Fornire all'indirizzario l'autorizzazione *RX per gli utenti individuali o per gruppi di utenti che hanno bisogno di accedere ai file SSL.

   Nota: non è possibile negare l'accesso ai file SSL ad utenti che hanno l'autorizzazione speciale *ALLOBJ.

Richiamo e configurazione dei certificati server

Prima di richiamare e configurare il certificato server, è necessario installare i prodotti che seguono:

- Programma su licenza IBM HTTP Server per iSeries  (5722-DG1)
- Opzione 34 sistema operativo di base (Digital Certificate Manager)

Il processo che si segue per richiamare e configurare il certificato server dipende dal tipo di certificato che si utilizza:

- Se si richiama un certificato da un'autorità garantita (quale VeriSign, Inc. o RSA Data Security, Inc.), installare il certificato su iSeries quindi applicarlo ai server host.
- Se si sceglie di non utilizzare un certificato da un'autorità garantita, è possibile creare il certificato da utilizzare su iSeries. Creare il certificato utilizzando Digital Certificate Manager:
  1. Creare l'autorizzazione del certificato sul server iSeries. Consultare l'argomento dell'Information Center, Agire come AC.
  2. Creare un certificato di sistema dall'autorizzazione del certificato creato.
  3. Assegnare quali server host utilizzeranno il certificato di sistema creato.

Utilizzo di un certificato da un'autorità garantita:

IBM Toolbox per Java fornisce un file keyring che supporta i certificati server da una serie di autorità garantite, rappresentate dalle società che seguono:

- IBM World Registry
- Integrion Financial Network
- RSA Data Security, Inc.
- Thawte Consulting
- VeriSign, Inc.

Il file keyring supporta già i certificati che l'utente ottiene da una di queste autorità garantite. Tutto ciò che si deve fare è richiamare i file zip che contengono gli algoritmi di codifica ed aggiungerli all'istruzione CLASSPATH.

Per utilizzare il certificato, completare le fasi che seguono:

1. Selezionare l'indirizzario in cui si desidera immettere i file zip.
2. Scaricare la versione di SSL che si desidera utilizzare copiando i file zip nell'indirizzario selezionato:
   - Per la codifica a 56 bit (utilizzata con il programma su licenza 5722-CE2), copiare /QIBM/ProdData/HTTP/Public/jt400/SSL56/sslightx.zip.
   - Per la codifica a 128 bit (utilizzata con il programma su licenza 5722-CE3), copiare /QIBM/ProdData/HTTP/Public/jt400/SSL128/sslightu.zip.
3. Aggiungere il file zip alla specifica CLASSPATH.

Utilizzo di un'autocertificazione:

Quando si decide di non utilizzare un certificato da un'autorità garantita, è necessario scaricare l'autocertificazione AC (Autorizzazione di certificazione) (da ogni server che dispone di un'autocertificazione AC) in modo che le classi di IBM Toolbox per Java possano utilizzarla. E' necessario inoltre richiamare i file zip che contengono gli algoritmi e aggiungerli all'istruzione CLASSPATH.

Per utilizzare l'autocertificazione, completare le fasi che seguono:

1. Selezionare l'indirizzario in cui si desidera immettere i file zip.
2. Scaricare la versione di SSL che si desidera utilizzare copiando gli algoritmi di codifica e i programmi di utilità necessari per gestire un'autocertificazione:
   - Per la codifica a 56 bit (utilizzata con i programmi su licenza 5722-CE2) copiare /QIBM/ProdData/HTTP/Public/jt400/SSL56/sslightx.zip, cfwk.zip, e ssltools.jar
   - Per la codifica a 128 bit (utilizzata con i programmi su licenza 5722-CE3) copiare /QIBM/ProdData/HTTP/Public/jt400/SSL128/sslightu.zip, cfwk.zip, e ssltools.jar.
3. Aggiungere ssltools.jar e i file zip all'istruzione CLASSPATH.
4. Creare un indirizzario nel client denominato <SSL>\com\ibm\as400\access dove <SSL> è l'indirizzario in cui sono stati copiati i file jar e zip.
5. Da una richiesta comandi nell'indirizzario <SSL> sul client, eseguire questo comando:

   ```
   java utilities.KeyringDB com.ibm.as400.access.KeyRing -connect <systemname>:<port>
   ```

   dove <port> è la porta server di un qualsiasi server host. Ad esempio, è possibile utilizzare 9476, che è la porta predefinita per il server di collegamento su iSeries.
6. Immettere il numero del certificato AC (Autorità di certificazione) che si desidera aggiungere al keyring. Assicurarsi di aggiungere il certificato AC e non il certificato sito
7. Quando si richiede di immettere un nome del certificato, è possibile immettere una stringa alfanumerica.

Nota: è necessario eseguire KeyringDB per ogni server che disponga di autocertificazione per aggiungere ogni certificato alla classe KeyRing. Su ogni iSeries in cui si desidera utilizzare collegamenti SSL, eseguire il seguente comando per aggiungere i certificati:

```
java utilities.KeyringDB com.ibm.as400.access.KeyRing -connect <systemname>:<port>
```

Dopo aver completato le fasi precedenti, l'impostazione delle autocertificazioni è terminata. E' possibile eseguire l'applicazione, dopo essersi assicurati che quanto segue si trova nell'istruzione CLASSPATH:

- l'indirizzario che contiene com\ibm\as400\access\KeyRing.class
- jt400.jar
- sslightx.zip o sslightu.zip (a seconda di quale file è stato scaricato)

Dal momento che jt400.jar contiene la copia predefinita di KeyRing.class, l'indirizzario che contiene com\ibm\as400\access\KeyRing.class deve trovarsi in CLASSPATH prima di jt400.jar.

Nota: invece di aggiungere l'indirizzario che contiene il file KeyRing.class all'istruzione CLASSPATH, è possibile sostituire la vecchia classe in jt400.jar con la nuova KeyRing.class.

Utilizzo di SSL per codificare i dati tra il client proxy e il server proxy:

E' possibile utilizzare SSL per codificare i dati scambiati tra il client proxy e il server proxy. I file forniti con il programma su licenza IBM iSeries Client Encryption (5722-CE2 o 5722-CE3) vengono utilizzati per codificare dati. Così come IBM Toolbox per Java, questi file sono indipendenti dalla piattaforma delle classi Java che abilitano il client proxy e il server proxy ad essere eseguiti su qualsiasi piattaforma con un JVM (Java virtual machine).

Eseguire le attività riportate di seguito per codificare i dati che si trovano tra il client proxy e il server proxy:

1. Impostare il server proxy per gestire i dati codificati.
2. Impostare il client proxy per gestire i dati codificati.
3. Utilizzare l'oggetto SecureAS400 per forzare IBM Toolbox per Java a codificare i dati.

Nota: i primi due passi creano solo un percorso protetto tra client proxy e server proxy. L'applicazione deve utilizzare l'oggetto SecureAS400 per indicare a IBM Toolbox per Java di far circolare i dati attraverso il percorso sicuro.L'utilizzo di un oggetto AS400 non codifica i dati, al contrario utilizza il percorso normale verso il server.

Se si desidera codificare dati che si trovano tra il client proxy e il server proxy, sono necessarie solo le classi Java che vengono fornite con il programma su licenza Client Encryption (5722-CE2 o 5722-CE3). Inoltre, se si desidera codificare i dati che si trovano tra il server proxy e i server iSeries, è necessario impostare la codifica tra il server proxy e il server iSeries .

Impostazione di SSL sul server proxy:

Per abilitare SSL, il server proxy deve disporre di un certificato server. Utilizzare la GUI IKeyman per creare il certificato server che il server proxy utilizzerà. IKeyman è uno strumenti della GUI, di conseguenza è necessario eseguirlo da una macchina client. Una volta creato il certificato, è possibile copiarlo su iSeries se il server proxy è in esecuzione su iSeries.

Per impostare il server proxy per la gestione dei dati codificati, eseguire le attività che seguono:

1. Impostare il client per eseguire la GUI IKeyman.
2. Creare un certificato server per il server proxy.
3. Avviare il server proxy utilizzando il certificato appena creato.

Impostazione del client per eseguire la GUI IKeyman

La GUI IKeyman è un programma Java che si basa sulle interfacce Java Swing 1.1. Per utilizzare IKeyman, sul client deve essere in esecuzione una JVM J2SE (Java 2 Standard Edition) supportata.

La GUI IKeyman è parte del programma su licenza IBM iSeries Client Encryption (5722-CE2 o 5722-CE3) in ssltools.jar. La procedura utilizzata per impostare il client per utilizzare SSL (e per eseguire IKeyman) dipende da quale versione del programma su licenza si sta eseguendo.

Impostare il client per utilizzare SSL completando le fasi che seguono:

1. Selezionare l'indirizzario sulla stazione di lavoro in cui si desidera immettere i file jar e zip necessari.
2. Copiare i file necessari nell'indirizzario selezionato:
   - Quando si utilizza la codifica a 56 bit, dopo aver caricato il programma su licenza 5722-CE2 su iSeries, copiare i file che seguono sulla stazione di lavoro:
     - /QIBM/ProdData/http/public/jt400/ssl56/sslightx.zip
     - /QIBM/ProdData/http/public/jt400/ssl56/ssltools.jar
     - /QIBM/ProdData/http/public/jt400/ssl56/cfwk.zip
     - /QIBM/ProdData/http/public/jt400/ssl56/cfwk.sec
   - Quando si utilizza la codifica a 128 bit, dopo aver caricato il programma su licenza 5722-CE3 su iSeries, copiare i file che seguono sulla stazione di lavoro:
     - /QIBM/ProdData/http/public/jt400/ssl128/sslightu.zip
     - /QIBM/ProdData/http/public/jt400/ssl128/ssltools.jar
     - /QIBM/ProdData/http/public/jt400/ssl128/cfwk.zip
     - /QIBM/ProdData/http/public/jt400/ssl128/cfwk.sec
3. Aggiungere il jar e i file zip alla specifica CLASSPATH. Non aggiungere .sec a CLASSPATH.

   Nota: cfwk.zip deve essere il primo elemento nel classpath.

Creazione di un certificato server

Utilizzare la GUI IKeyman per creare un'autocertificazione.

Nota: se la GUI IKeyman interrompe l'esecuzione, accertarsi che cfwk.zip sia il primo elemento in CLASSPATH e che cfwk.sec si trovi nello stesso indirizzario di cfwk.zip.

Creare un certificato server per il server proxy completando le fasi che seguono:

1. Avviare la GUI IKeyman utilizzando il comando che segue:
   ```
   java -Dkeyman.javaOnly=true com.ibm.gsk.ikeyman.Ikeyman
   ```
2. Dal menu **File database chiave** IKeyman, selezionare **Nuovo**.
3. Nella finestra di dialogo **Nuovo**, non modificare **Tipo database chiave**, che dovrebbe essere la **classe database chiave SSLight**.
4. Immettere il **Nome file** (ad esempio, `ProxyServerKeyring.class`) o fare clic su **Sfoglia** per collocare il file classe che si desidera utilizzare per il keyring.

   Nota: tenere a mente il nome del file keyring, poiché sarà necessario per avviare il server proxy sicuro.
5. Immettere una **Posizione** (percorso) o accettare la posizione predefinita, che è l'indirizzario di lavoro corrente, quindi fare clic su **OK**.
6. Nella finestra di dialogo di dialogo **Richiesta parola d'ordine**, immettere una **Parola d'ordine** e una **Parola d'ordine di conferma**, quindi fare clic su **OK**.(**Imposta il tempo di scadenza** è facoltativo e non è necessario selezionarlo).

Nota: tenere a mente la parola d'ordine, che sarà necessaria per avviare il server proxy sicuro. Le icone chiave in questa casella di dialogo rappresentano una sicurezza relativa della parola d'ordine. Una maggiore sicurezza della parola d'ordine implica un misto di caratteri alfanumerici in maiuscolo o minuscolo.

7. Dal menu file **Crea** di IKeyman, selezionare **Nuova autocertificazione**.
8. Nella finestra di dialogo **Crea nuova autocertificazione**, immettere un'**etichetta chiave** (ad esempio, `MyCertificate`) e **Organizzazione**.
9. Fare clic sull'elenco **Paese** selezionare un paese o una regione, immettere un **Periodo di validità** o accettare il valore predefinito, quindi fare clic su **OK**.
10. Dal menu **File database chiave** IKeyman, selezionare **Chiudi**, quindi (dallo stesso menu) fare clic su **Esci**.

E' possibile visualizzare il keyring appena creato nell'indirizzario corrente.

Avvio del server proxy utilizzando il nuovo certificato

Prima di avviare il server proxy, assicurarsi che CLASSPATH per il server proxy contenga jt400.jar, sslightx.zip e l'ubicazione del server proxy keyring.

Avviare il server proxy utilizzando il certificato creato. Utilizzare i parametri -keyringName e -keyringPassword per passare queste informazioni al server proxy. Ad esempio:

```
java com.ibm.as400.access.ProxyServer -keyringName ProxyServerKeyring -keyringPassword pxypswrd
```

Impostazione di SSL sul client proxy:

La procedura che segue porta all'aggiunta del certificato server nel database certificato sul client, memorizzato in un file .class Java. L'aggiunta del certificato server al client è necessaria poiché il server utilizza un'autocertificazione.

Impostare il client proxy per scambiare dati codificati completando le attività che seguono:

1. Impostare il server proxy per gestire dati codificati, quindi avviare il server proxy.
2. Impostare il client per utilizzare SSL.
3. Utilizzare KeyringDB per richiamare il certificato server sul server proxy.
4. Impostare il client per utilizzare il file aggiornato KeyRing.class.
5. Impostare le impostazioni proxy sicuro sul client.

Impostazione del client per utilizzare SSL

Lo strumento che scarica il certificato (KeyringDB) è un programma Java. KeyringDB è una parte del programma su licenza IBM iSeries Client Encryption (5722-CE2 o 5722-CE3) in ssltools.jar. La procedura utilizzata per impostare il client per utilizzare SSL dipende da quale versione del programma su licenza si sta eseguendo.

Dopo aver impostato il server proxy, impostare il client per utilizzare SSL completando le fasi che seguono:

1. Selezionare l'indirizzario sulla stazione di lavoro in cui si desidera immettere i file jar e zip necessari.
2. Copiare i file necessari nell'indirizzario selezionato:
   - Quando si utilizza la codifica a 56 bit, dopo aver caricato il programma su licenza 5722-CE2 su iSeries, copiare i file che seguono sulla stazione di lavoro:
     - /QIBM/ProdData/http/public/jt400/ssl56/sslightx.zip
     - /QIBM/ProdData/http/public/jt400/ssl56/ssltools.jar
     - /QIBM/ProdData/http/public/jt400/ssl56/cfwk.zip

- Quando si utilizza la codifica a 128 bit, dopo aver caricato il programma su licenza 5722-CE3 sul server, copiare i file che seguono sulla stazione di lavoro:
  - /QIBM/ProdData/http/public/jt400/ssl128/sslightu.zip
  - /QIBM/ProdData/http/public/jt400/ssl128/ssltools.jar
  - /QIBM/ProdData/http/public/jt400/ssl128/cfwk.zip

3. Aggiungere il jar e i file zip all'istruzione CLASSPATH.
4. Creare un indirizzario nel client denominato <SSL>\com\ibm\as400\access dove <SSL> è l'indirizzario in cui sono stati copiati i file jar e zip.

Aggiunta del certificato server per il server proxy

KeyringDB crea un nuovo file KeyRing.class che contiene il certificato server e lo immette nel sottoindirizzario com\ibm\as400\access fuori dall'indirizzario corrente.

Utilizzare lo strumento KeyringDB per aggiungere il certificato server a KeyRing.class completando le fasi che seguono:

1. Dall'indirizzario in cui sono stati immessi i file jar e zip, eseguire il comando:

   ```
   java utilities.KeyringDB com.ibm.as400.access.KeyRing -connect proxyServerName:port
   ```

   dove:
   - *proxyServerName* è il nome della macchina su cui il server proxy è in esecuzione
   - *port* è la porta su cui è in ascolto il server proxy sicuro (3471 per impostazione predefinita)

   Ad esempio:

   ```
   java utilities.KeyringDB com.ibm.as400.access.KeyRing -connect myProxyServer:3471
   ```

2. Quando si richiede quale certificato utilizzare, scegliere il certificato sito 0.
3. Quando si richiede di immettere un nome del certificato, è possibile immettere una stringa alfanumerica.

Impostazione del client per utilizzare il file KeyRing.class aggiornato

Il file jt400Proxy.jar contiene KeyRing.class. Per impostare il client per utilizzare il file aggiornato KeyRing.class file, assicurarsi che quanto segue si trovi nell'istruzione specifica CLASSPATH:

- l'indirizzario che contiene com\ibm\as400\access\KeyRing.class
- jt400Proxy.jar
- sslightx.zip o sslightu.zip (a seconda di quale file è stato scaricato)
- cfwk.zip

Dal momento che jt400Proxy.jar contiene la copia predefinita di KeyRing.class, l'indirizzario che contiene com\ibm\as400\access\KeyRing.class deve trovarsi in CLASSPATH prima di jt400Proxy.jar.

Nota: invece di aggiungere l'indirizzario che contiene il file KeyRing.class all'istruzione CLASSPATH, è possibile aggiungere il nuovo KeyRing.class al file jt400Proxy.jar. L'aggiunta del nuovo file KeyRing.class a jt400Proxy.jar sostituisce la versione precedente.

Configurazione delle impostazioni per il proxy sicuro sul client

Per indicare al client proxy di comunicare con il server proxy attraverso un collegamento sicuro, impostare le proprietà di sistema che seguono:

```
com.ibm.as400.access.AS400.proxyServer=proxyServer
```

dove *proxyServer* è il nome della macchina su cui il server proxy è in esecuzione

```
com.ibm.as400.access.SecureAS400.proxyEncryptionMode=mode
```

dove *mode* è uno dei numeri interi che seguono:

- 1 per codificare tra il client proxy e il server proxy
- 2 per codificare tra il server proxy e il server iSeries
- 3 per codificare tra il client proxy e il server proxy e tra il server proxy e il server iSeries

Ad esempio, i comandi che seguono avviano un'applicazione utilizzando SSL:

```
java -Dcom.ibm.as400.access.AS400.proxyServer=myProxyServer
      -Dcom.ibm.as400.access.SecureAS400.proxyEncryptionMode=1   myApplication
```

Servizi di autenticazione

Le classi sono fornite dall'IBM Toolbox per Java che interagisce con i servizi di sicurezza forniti da OS/400. Specificamente, il supporto viene fornito per autenticare l'identità di un utente, qualche volta indicato come *principal*, e per la parola d'ordine rispetto al registro utenti OS/400. Può essere quindi stabilita una credenziale che rappresenta l'utente autenticato. E' possibile utilizzare la credenziale per modificare l'identità del sottoprocesso OS/400 corrente in modo da eseguire funzioni con le autorizzazioni e i permessi dell'utente autenticato. In effetti, questo scambio di identità fa sì che il sottoprocesso agisca come se fosse stato eseguito un collegamento dall'utente autenticato.

Nota: i servizi per stabilire e scambiare le credenziali sono supportati soltanto su server al livello del release V5R1M0 o superiore.

Panoramica del supporto fornito

L'oggetto AS400 fornisce l'autenticazione per un profilo utente dato e una parola d'ordine rispetto al server. E' possibile anche richiamare certificati e token di profilo Kerberos che rappresentano profili utente e parole d'ordine autenticati per il sistema.

Nota: l'utilizzo di certificati Kerberos richiede di installare J2SDK, v1.4 e configurare l'API (Application Programming Interface) JGSS (Java General Security Services). Per ulteriori informazioni su JGSS, consultare J2SDK, v1.4 Security Documentation  .

Per utilizzare certificati Kerberos, impostare solo il nome di sistema (e non la parola d'ordine) nell'oggetto AS400. L'identità dell'utente viene richiamata attraverso la framework JGSS. In un oggetto AS400 è possibile impostare un mezzo di autenticazione alla volta. L'impostazione della parola d'ordine eliminerà il contenuto di ogni ticket Kerberos o token del profilo.

Per utilizzare i token di profilo, utilizzare il metodo getProfileToken() per richiamare istanze della classe ProfileTokenCredential.Considerare i token di profilo come una rappresentazione di un profilo utente e della parola d'ordine autenticati per un server specifico. I token di profilo scadono in base al tempo, fino ad un'ora, ma possono essere aggiornati in alcuni casi per fornire una durata prolungata.

Nota: se si utilizza la classe ProfileTokenCredential, accertarsi di esaminare le informazioni alla fine di questa pagina che discute i metodi per l'impostazione dei token.

Il seguente esempio crea un oggetto di sistema ed utilizza tale oggetto per creare un token di profilo. L'esempio utilizza inoltre il token di profilo per creare un altro oggetto di sistema e utilizza il secondo oggetto di sistema per collegarsi al servizio di comando:

```
AS400 system = new AS400("mySystemName", "MYUSERID", "MYPASSWORD");
ProfileTokenCredential myPT = system.getProfileToken();
AS400 system2 = new AS400("mySystemName", myPT);
system2.connectService(AS400.COMMAND);
```

Impostazione delle identità del sottoprocesso

E' possibile stabilire una credenziale sia in un contesto remoto che locale. Una volta creata, è possibile serializzare o distribuire la credenziale come necessaria tramite l'applicazione di chiamata. Una volta passata ad un processo in esecuzione sul server associato, una credenziale può essere utilizzata per modificare o *scambiare* l'identità del sottoprocesso OS/400 ed eseguire funzioni per conto dell'utente precedentemente autenticato.

Un'applicazione pratica di questo supporto la si può trovare in un'applicazione a due livelli, con l'autenticazione di un profilo utente e della parola d'ordine eseguita tramite una GUI al primo livello (ad esempio un PC) e le funzioni eseguite per l'utente in questione al secondo livello (il server). Utilizzando ProfileTokenCredentials, l'applicazione può evitare di passare direttamente gli ID utente e le parole d'ordine sulla rete. Il token di profilo può essere quindi distribuito al programma al secondo livello, che può eseguire lo *swap()* e funzionare sotto le autorizzazioni e i permessi OS/400 assegnati all'utente.

Nota: anche se intrinsecamente più sicuri del passare un profilo utente e una parola d'ordine grazie alla durata limitata, i token profilo dovrebbero ancora essere considerati informazioni critiche dall'applicazione e gestiti di conseguenza. Dal momento che il token rappresenta un utente e una parola d'ordine autenticati, può essere potenzialmente sfruttato da un'applicazione potenzialmente dannosa per eseguire funzioni per conto dell'utente. E' responsabilità dell'applicazione assicurare l'accesso sicuro alle credenziali.

Metodi per l'impostazione di token in ProfileTokenCredential

I metodi per l'impostazione di token nella classe ProfileTokenCredential richiedono che l'utente distingua i modi differenti di specificare le parole d'ordine:

- Come valore speciale, come ad esempio *NOPWD o *NOPWDCHK, utilizzando un numero intero valore speciale definito
- Come parola d'ordine per il profilo utente utilizzando una Stringa che rappresenta la parola d'ordine

Nota: in V5R3, IBM Toolbox per Java scarta i metodi setToken che non richiedono all'utente di distinguere la modalità di specifica della parola d'ordine.

Inoltre, i metodi setToken consentono ad utenti remoti di specificare valori speciali di parola d'ordine e consentono parole d'ordine profilo utente più lunghe fino a 128 caratteri.

Per specificare un numero intero valore speciale parola d'ordine, come ad esempio *NOPWD o *NOPWDCHK, utilizzare uno dei seguenti metodi:

- setToken(AS400Principal principal, int passwordSpecialValue)
- setToken(String name, int passwordSpecialValue)

La classe ProfileTokenCredential include le seguenti costanti statiche per numeri interi valori speciali parola d'ordine:

- ProfileTokenCredential.PW_NOPWD: indica *NOPWD
- ProfileTokenCredential.PW_NOPWDCHK: indica *NOPWDCHK

Per specificare una parola d'ordine profilo utente come Stringa, utilizzare uno dei seguenti metodi:

- setTokenExtended(AS400Principal principal, String password)
- setTokenExtended(String name, String password)

I metodi setTokenExended non consentono di passare stringhe valori speciali parola d'ordine come parametro della parola d'ordine. Ad esempio, questi metodi non consentono una stringa parola d'ordine di *NOPWD.

Per ulteriori informazioni, consultare le seguenti informazioni di riferimento Javadoc:

ProfileTokenCredential

Esempio

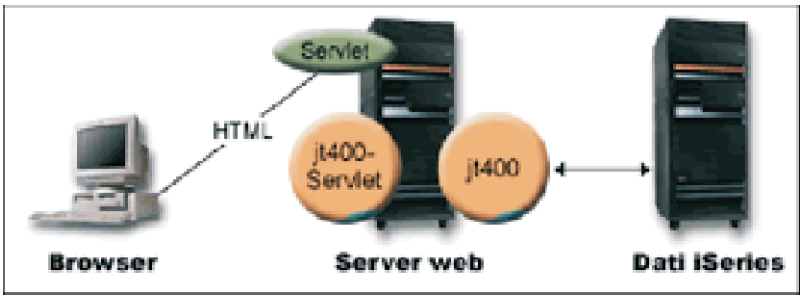
Fare riferimento a questo codice per un esempio di come utilizzare la credenziale del token di profilo per scambiare l'identità del sottoprocesso OS/400 ed eseguire le funzioni per conto di un utente specifico.

Classi servlet

Le classi servlet fornite con IBM Toolbox per Java gestiscono le classi access, ubicate sul server web, per fornire accesso alle informazioni che si trovano sul server iSeries.E' possibile decidere come utilizzare le classi servlet per assistenza nei propri progetti servlet.

Il diagramma che segue mostra come le classi servlet operano tra i dati del browser, del server web e di dati iSeries. Un browser si collega al server web su cui è in esecuzione il servlet. I file jt400Servlet.jar e jt400.jar risiedono sul server web poiché le classi servlet utilizzano alcune delle classi access per richiamare i dati e le classi HTML per presentarli. Il server web è collegato al server iSeries in cui si trovano i dati.

Figura 1: come funzionano i servlet



"Descrizione lunga della Figura 1: come funzionano i servlet (rzahh585.gif)"

Esistono quattro tipi di classi servlet incluse in IBM Toolbox per Java:

- Classi Authentication
- Classi RowData
- Classi RowMetaData
- Classi Converter

Nota: il file jt400Servlet.jar include sia le classi HTML che Servlet. E' necessario aggiornare CLASSPATH in modo che punti sia a jt400Servlet.jar che a jt400.jar se si vogliono utilizzare classi nei pacchetti com.ibm.as400.util.html e com.ibm.as400.util.servlet.

Per ulteriori informazioni relative ai servlet in generale, consultare la sezione riferimenti.

Descrizione lunga della Figura 1: come funzionano i servlet (rzahh585.gif)

In IBM Toolbox per Java: classi Servlet

Questa figura illustra in modo generale il funzionamento dei servlet.

Descrizione

La figura è composta da quanto segue:

- Un'immagine sulla sinistra di un personal computer, etichettata 'Browser', che rappresenta un'istanza di un browser in esecuzione su un personal elaborazione computer.
- Un'immagine di un server iSeries sulla destra, etichettata 'Dati iSeries', che rappresenta l'ubicazione dei dati ai quali si desidera che il servlet acceda.
- Un'immagine di un server iSeries nel mezzo (tra le altre due immagini), etichettata 'Server web', che rappresenta il server Web. Diverse figure etichettate nell'immagine Server web indicano file o funzioni che risiedono nel Server web:
  - Un ovale verde etichettato Servlet che rappresenta l'ubicazione del codice servlet.
  - Un cerchio rossiccio etichettato jt400Servlet che indica l'ubicazione del file jt400Servlet.jar.
  - Un cerchio rossiccio etichettato jt400 che indica l'ubicazione del file jt400.jar.

  Nota: il Server Web non deve trovarsi su un server iSeries, ma questo può accadere e può anche essere lo stesso server indicato dall'immagine Dati iSeries.
- Linee che collegano le immagini.

Una linea etichettata HTML collega il Browser (l'immagine di sinistra) ad un Servlet (l'ovale verde) sul Server web (l'immagine centrale). La linea è etichettata HTML perché i servlet il più delle volte utilizzano HTML per 'servire' dati al browser.

Il Server Web sta elaborando due file jar di IBM Toolbox per Java (i cerchi rossicci), jt400Servlet.jar e jt400.jar. Le classi in jt400Servlet.jar, insieme alle classi in jt400.jar, abilitano il Server web ad eseguire un servlet che si collega facilmente ai server che contengono Dati iSeries (l'immagine di destra). La linea con le frecce ad entrambi gli estremi che collega le due immagini indica il suddetto collegamento.

Classi Authentication

Due classi nel pacchetto servlet eseguono autenticazioni per i servlet: AuthenticationServlet e AS400Servlet.

Classe AuthenticationServlet

AuthenticationServlet è una implementazione di HttpServlet che esegue funzioni di autenticazione di base per i servlet. Sottoclassi di AuthenticationServlet sostituiscono uno o più dei seguenti metodi:

- Sostituire il metodo validateAuthority() per eseguire l'autenticazione (obbligatorio)
- Sostituire il metodo bypassAuthentication() in modo che la sottoclasse autentichi solo alcune richieste
- Sostituire il metodo postValidation() per consentire un'elaborazione ulteriore della richiesta dopo l'autenticazione

La classe AuthenticationServlet fornisce metodi che consentono di:

- Inizializzare il servlet
- Richiamare l'ID utente autenticato
- Impostare un ID utente dopo aver ignorato l'autenticazione
- Registrare leeccezioni e imessaggi

Classe AS400Servlet

La classe AS400Servlet è una sottoclasse astratta di AuthenticationServlet che rappresenta un servlet HTML. E' possibile utilizzare un lotto di collegamenti per condividere collegamenti e gestire il numero di collegamenti al server che un utente servlet può avere.

La classe AS400Servlet fornisce metodi che consentono di:
- Convalidare l'autorizzazione dell'utente (sostituendo il metodo validateAuthority() della classe AuthenticationServlet)
- Collegarsi ad un sistema
- Richiamare e restituire oggetti del lotto di collegamenti al e dal lotto.
- Chiudere il lotto di collegamenti
- Richiamare e impostare le tag iniziali del documento HTML
- Richiamare e impostare le tag finali del documento HTML

Per ulteriori informazioni relative ai servlet in generale, consultare la sezione riferimenti.

Classe RowData

La classe RowData è una classe astratta che fornisce un metodo per descrivere e accedere ad un elenco di dati.

Esistono quattro classi principali che estendono la classe RowData:
- ListRowData
- RecordListRowData
- ResourceListRowData
- SQLResultSetRowData

Le classi RowData consentono di:
- Richiamare e impostare la posizione corrente
- Richiamare i dati riga in una determinata colonna utilizzando il metodo getObject()
- Richiamare i meta dati per la riga
- Richiamare o impostare le proprietà per un oggetto su una colonna data
- Richiamare il numero di righe nell'elenco utilizzando il metodo length().

Posizione RowData

Esistono diversi metodi che consentono di richiamare e impostare la posizione corrente in un elenco. La tabella che segue elenca i metodi di impostazione e di richiamo per le classi RowData.

Metodi Set		**Metodi Get**
absolute()	next()	getCurrentPosition()
afterLast()	previous()	isAfterLast()
beforeFirst()	relative()	isBeforeFirst()
first()		isFirst()
last()		isLast()

Classe ListRowData:

La classe ListRowData consente di effettuare le seguenti operazioni:
- Aggiungere e rimuovere le righe a/da un elenco di risultati.
- Richiamare ed impostare la riga
- Richiamare informazioni sulle colonne dell'elenco con il metodo getMetaData()
- Impostare le informazioni relative alla colonna con il metodo setMetaData()

La classe ListRowData rappresenta un elenco di dati. ListRowData può rappresentare vari tipi di informazioni, incluse le seguenti, mediante IBM Toolbox per Java "Classi Access" a pagina 23:

- Un indirizzario nell'IFS (Integrated File System)
- Un elenco di lavori
- Un elenco di messaggi in una coda messaggi
- Un elenco di utenti
- Un elenco di stampanti
- Un elenco di file di spool

Esempio

Il seguente esempio mostra le funzione delle classi ListRowData e HTMLTableConverter. L'esempio mostra il codice Java, il codice HTML e l'aspetto dell'HTML.

"Esempio: utilizzo di ListRowData" a pagina 647

Classe RecordListRowData:

La classe RecordListRowData consente di effettuare le seguenti operazioni:

- Aggiungere ed eliminare righe nell'/dall'elenco di record.
- Richiamare ed impostare la riga
- Impostare il formato record con il metodo setRecordFormat.
- Richiamare il formato record.

La classe RecordListRowData rappresenta un elenco record. E' possibile ottenere un record dal server in formati diversi, compresi:

- Un record da scrivere o leggere da un file del server
- Una voce in una coda di dati
- I dati del parametro da una chiamata al programma
- Qualsiasi dato restituito che necessita di conversione tra il formato del server e il formato Java

Questo esempio illustra come funzionano RecordListRowData e HTMLTableConverter. Mostra il codice java, il codice HTML e l'aspetto dell'HTML.

Classe ResourceListRowData:

La classe ResourceListRowData rappresenta un elenco di risorse di dati. Utilizzare gli oggetti ResourceListRowData per rappresentare qualsiasi implementazione dell'interfaccia ResourceList.

Gli elenchi di risorse vengono formattati in una serie di righe, ognuna delle quali contiene un numero finito di colonne determinato dal numero di ID dell'attributo colonna. Ogni colonna all'interno di una riga contiene una singola voce dati.

La classe ResourceListRowData offre dei metodi che consentono di eseguire queste operazioni:

- Richiamare ed impostare gli ID dell'attributo colonna
- Richiamare ed impostare l'elenco di risorse
- Richiamare il numero di righe nell'elenco
- Richiamare i dati colonna per la riga corrente
- Richiamare l'elenco di proprietà dell'oggetto dati
- Richiamare i metadati per l'elenco

Esempio: presentazione di un elenco di risorse in un servlet

Esonero di responsabilità per gli esempi di codice

Il seguente esonero di responsabilità si applica a tutti gli esempi IBM Toolbox per Java:

> L'IBM fornisce una licenza non esclusiva per utilizzare tutti gli esempi del codice di programmazione da cui creare funzioni simili personalizzate, in base a richieste specifiche.
>
> Questo codice di esempio è fornito dall'IBM con la sola funzione illustrativa. Questi esempi non sono stati interamente testati in tutte le condizioni. IBM, perciò, non fornisce nessun tipo di garanzia o affidabilità implicita, rispetto alla funzionalità o alle funzioni di questi programmi.
>
> Tutti i programmi qui contenuti vengono forniti all'utente ″COSI' COME SONO″ senza garanzie di alcun tipo. Le garanzie implicite di non contraffazione, commerciabilità e adeguatezza a scopi specifici sono espressamente vietate.

Classe SQLResultSetRowData:

La classe SQLResultSetRowData rappresenta un insieme di risultati SQL come unelenco di dati. Questi dati vengono creati da una istruzione SQL attraverso JDBC.Assieme ai metodi forniti, è possibilerichiamare e impostare l'insieme di risultati meta dati.

Questo esempio mostra come funzionano ListRowData e HTMLTableConverter.Mostra il codice java, il codice HTML e l'aspetto dell'HTML.

Classi RowMetaData

La classe RowMetaData definisce un'interfaccia che si utilizza per trovare informazioni relative alle colonne di un oggetto RowData.

Con le classi RowMetaData è possibile effettuare le seguenti operazioni:

- Richiamare il numero di colonne
- Richiamare il nome, il tipo o la dimensione della colonna
- Richiamare o impostare l'etichetta della colonna
- Richiamare la precisione o la scala dei dati della colonna
- Determinare se i dati della colonna sianodati di testo

Esistono tre classi principali che implementano la classe RowMetaData. Queste classi forniscono tutte le funzioni RowMetaData elencate precedentemente oltre ad avere proprie funzioni specifiche:

- ListMetaData
- RecordFormatMetaData
- SQLResultSetMetaData

Classe ListMetaData:

La classe ListMetaData consente di consultare informazioni e di modificare le impostazioni relative alle colonne in "Classe ListRowData" a pagina 238. Essa utilizza il metodo setColumns() per impostare il numero di colonne, eliminando il contenuto di tutte le precedenti informazioni sulla colonna. In alternativa, è possibile anche immettere il numero di colonne quando si impostano i parametri del programma di creazione.

Esempio

Il seguente esempio mostra le funzione di ListMetaData, ListRowData e HTMLTableConverter. Mostra il codice Java, il codice HTML, e l'aspetto dell'HTML.

"Esempio: utilizzo di ListRowData" a pagina 647

Classe RecordFormatMetaData:

RecordFormatMetaData utilizza la classe RecordFormat IBM Toolbox per Java. Consente di fornire il formato record quando si impostano i parametri del programma di creazione o si utilizzano i metodi get e set per accedere al formato record.

Il seguente esempio mostra come creare un oggetto RecordFormatMetaData:

```
// Creare un oggetto RecordFormatMetaData da un formato record del file sequenziale.
RecordFormat recordFormat = sequentialFile.getRecordFormat();
RecordFormatMetaData metadata = new RecordFormatMetaData(recordFormat);

// Visualizzare i nomi colonna del file.
int numberOfColumns = metadata.getColumnCount();
for (int column=0; column < numberOfColumns; column++)
{
      System.out.println(metadata.getColumnName(column));
}
```

Classe SQLResultSetMetaData:

La classe SQLResultSetMetaData restituisce informazioni relative alle colonne di un oggetto SQLResultSetRowData.E' possibile fornire la serie di risultati quando si impostano i parametri del programma creazione e utilizzare i metodi get e set per accedere all'insieme di risultati metadati.

L'esempio che segue mostra come creare un oggetto SQLResultSetMetaData:

```
// Creare un oggetto SQLResultSetMetaData dai metadati della serie di risultati.
SQLResultSetRowData rowdata = new SQLResultSetRowData(resultSet);
SQLResultSetMetaData sqlMetadata  = rowdata.getMetaData();

// Visualizzare la precisione colonna per le colonne non testo.
String name = null;
int numberOfColumns = sqlMetadata.getColumnCount();
for (int column=0; column < numberOfColumns; column++)
{
      name = sqlMetadata.getColumnName(column);
      if (sqlMetadata.isTextData(column))
   {
            System.out.println("Column: " + name + " contains text data.");
}
                        else
   {
            System.out.println("Column: " + name + " has a precision of " + sqlMetadata.getPrecision(column));
   }
}
```

Classi Converter

Si utilizzano le classi Converter per convertire dati di riga in schiere di stringhe formattate. Il risultato è in formato HTML ed è pronto per la presentazione sulla pagina HTML. Le seguenti classi eseguono la conversione per conto dell'utente:

- StringConverter
- HTMLFormConverter
- HTMLTableConverter

Classe StringConverter:

La classe StringConverter è una classe astratta che rappresenta un programma di conversione stringa dati riga.Fornisce un metodo convert() per convertire dati riga.Ciò restituisce una rappresentazione della schiera stringa dati della riga.

Classe HTMLFormConverter:

La classe HTMLFormConverter estende StringConverter fornendo un metodo di conversione aggiuntivo, denominato convertToForms().Questo metodo converte i dati riga in una schiera di tabelle HTML a riga singola. E' possibile utilizzare queste tag della tabella per visualizzare le informazioni formattate in un browser.

E' possibile adattare l'aspetto del modulo HTML utilizzando i vari metodi di richiamo e impostazione per visualizzare o modificare gli attributi del modulo. Ad esempio, alcuni degli attributi che è possibile impostare includono:

- Allineamento
- Spaziatura cella
- Hyperlink di intestazione
- Ampiezza

Esempio: utilizzo di HTMLFormConverter

Il seguente esempio mostra l'utilizzo di HTMLFormConverter. E' possibile compilare ed eseguire questo esempio con un server web in esecuzione.

Utilizzare HTMLFormConverter

Classe HTMLTableConverter:

La classe HTMLTableConverter estende StringConverter fornendo un metodoconvertToTables().Questo metodo converte i dati di riga in una schiera di tabelle HTML che il servlet può utilizzare per visualizzare l'elenco su un browser.

E' possibile utilizzare i metodi getTable() e setTable() per scegliere una tabella predefinita che sarà utilizzata durante la conversione.E' possibile impostare intestazioni della tabella nell'oggetto tabella HTML o utilizzare i meta dati per le informazioni di intestazione impostando setUseMetaData() su vero.

Il metodo setMaximumTableSize() consente di limitare il numero di righe in una tabella singola.Se i dati di riga non si adattano alla dimensione specificata della tabella, il convertitore produrrà un altro oggetto tabella HTML nella schiera di emissione. Ciò continuerà fino a quando tutti i dati di riga saranno convertiti.

Esempi

Gli esempi che seguono illustrano come utilizzare la classe HTMLTableConverter:

- Esempio: utilizzo di ListRowData
- Esempio: utilizzo di RecordListRowData
- Esempio: utilizzo di SQLResultSetRowData
- Esempio: presentazione di ResourceList in un servlet

Esonero di responsabilità per gli esempi di codice

Il seguente esonero di responsabilità si applica a tutti gli esempi IBM Toolbox per Java:

L'IBM fornisce una licenza non esclusiva per utilizzare tutti gli esempi del codice di programmazione da cui creare funzioni simili personalizzate, in base a richieste specifiche.

Questo codice di esempio è fornito dall'IBM con la sola funzione illustrativa. Questi esempi non sono stati interamente testati in tutte le condizioni. IBM, perciò, non fornisce nessun tipo di garanzia o affidabilità implicita, rispetto alla funzionalità o alle funzioni di questi programmi.

Tutti i programmi qui contenuti vengono forniti all'utente ″COSI' COME SONO″ senza garanzie di alcun tipo. Le garanzie implicite di non contraffazione, commerciabilità e adeguatezza a scopi specifici sono espressamente vietate.

Classi Utility

Le classi programma di utilità consentono di eseguire attività specifiche. IBM Toolbox per Java offre i seguenti programmi di utilità:

- AS400ToolboxInstaller: Consente di installare e aggiornare le classi IBM Toolbox per Java sul client.Questa funzione è disponibile come programma Java e ha un'API (Application Programming Interface).
- AS400ToolboxJarMaker: Consente un caricamento più veloce del file JAR di IBM Toolbox per Java creando un file JAR più piccolo da uno più grande o decomprimendo in maniera selettiva un file JAR per ottenere l'accesso ai singoli file di contenuto.
- CommandHelpRetriever: richiama e crea testo di aiuto per i comandi CL (control language) OS/400.
- CommandPrompter: Richiede il parametro su un dato comando.CommandPrompter fornisce una funzionalità che è simile alla richiesta comandi CL di iSeries (premendo F4) ed uguale alla Richiesta comandi di Management Central.
- RunJavaApplication and VRunJavaApplication: Consentono di eseguire un programma Java su un server iSeries da una riga comandi.
- JPing: Consente di richiedere al server di determinare i servizi attivi.E' possibile inoltre specificare se si desidera effettuare il ping delle porte SSL.

Classi di installazione e aggiornamento client

Le classi IBM Toolbox per Java possono essere consultate nella loro ubicazione nell'IFS sul server. Dal momento che le PTF (program temporary fixes) vengono applicate in questa posizione, i programmi Java che accedono a queste classi direttamente sul server ricevono automaticamente questi aggiornamenti. L'accesso alle classi dal server non sempre funziona, soprattutto per le situazioni che seguono:

- Se un collegamento di comunicazione a bassa velocità collega il server e il client, l'esecuzione del caricamento delle classi dal server potrebbe non essere accettabile.
- Se la applicazioni Java utilizzano la variabile di ambiente CLASSPATH per accedere alle classi sul client del file system, è necessario iSeries Access per Windows per reindirizzare le chiamate del file system sul server. Potrebbe non essere possibile per iSeries Access per Windows risiedere nel client.

In questi casi, l'installazione delle classi sul client è la soluzione migliore. La classe AS400ToolboxInstaller fornisce funzioni di aggiornamento e di installazione client per gestire le classi IBM Toolbox per Java quando esse risiedono su un client.

Utilizzo di AS400ToolboxInstaller

L'oggetto AS400ToolboxInstaller è un programma e un'interfaccia programmabile. L'oggetto dispone di un metodo main() in modo da poter essere eseguito dalla riga comandi. Dispone inoltre di metodi public worker in modo da poter essere incluso e richiamato dall'applicazione.

L'oggetto AS400ToolboxInstaller può essere utilizzato sia per installare i file IBM Toolbox per Java sul client che per mantenerli aggiornati. Quando vengono utilizzati per la prima volta, i file IBM Toolbox per

Java vengono copiati dal server sulla stazione di lavoro. Quando si applicano le PTF al server, l'oggetto AS400ToolboxInstaller aggiorna i file sulla stazione di lavoro con i nuovi file sul server.

La classe AS400ToolboxInstaller copia i file sul file system locale del client. Questa classe potrebbe non funzionare in un'applet; molti browser non consentono al programma Java di scrivere sul file system locale.

Esecuzione della classe AS400ToolboxInstaller dalla riga comandi

La classe AS400ToolboxInstaller può essere utilizzata come programma autonomo, eseguito dalla riga comandi. L'esecuzione di AS400ToolboxInstaller dalla riga comandi implica che l'utente non deve scrivere un programma. Al contrario lo si avvia come applicazione Java per installare, disinstallare o aggiornare le classi IBM Toolbox per Java.

Specificando l' opzione di installazione, disinstallazione o di confronto appropriata si richiama la classe AS400ToolboxInstaller con i comandi che seguono:

```
java utilities.AS400ToolboxInstaller [options]
```

L'opzione **-source** indica dove possono essere trovate le classi IBM Toolbox per Java e **-target** indica dove devono essere memorizzate le classi IBM Toolbox per Java nel client.

Sono inoltre disponibili opzioni per installare l'intero IBM Toolbox per Java o solo alcune funzioni. Ad esempio, per installare o aggiornare classi nel pacchetto Access di IBM Toolbox per Java (jt400.jar) sulla propria stazione di lavoro, utilizzare il seguente comando:

```
java utilities.AS400ToolboxInstaller -install -package ACCESS -source myAS400    -target c:\toolbox
```

L'esempio precedente presume che c:\toolbox sia l'indirizzario che contiene i file jar di IBM Toolbox per Java.

Come incorporare la classe AS400ToolboxInstaller nel programma

La classe AS400ToolboxInstaller fornisce le API necessarie per installare, disinstallare e aggiornare le classi IBM Toolbox per Java all'interno del programma nel client.

Utilizzare i metodi install() per installare o aggiornare le classi IBM Toolbox per Java.Per l'installazione o l'aggiornamento, fornire il percorso sorgente e di destinazione e il nome del pacchetto delle classi nel programma Java. L'URL sorgente punta all'ubicazione dei file di controllo sul server. La struttura dell'indirizzario viene copiata dal server sul client.

Il metodo install() copia soltanto i file; **non** aggiorna la variabile di ambiente CLASSPATH.Se il metodo install() ha esito positivo, il programma Java può richiamare il metodo getClasspathAdditions() per determinare cosa deve essere aggiunto alla variabile di ambiente CLASSPATH.

L'esempio che segue mostra come utilizzare la classe AS400ToolboxInstaller per installare file da un server denominato ″mySystem″ nell'indirizzario ″jt400″ sull'unità d: e come determinare cosa deve essere aggiunto alla variabile d'ambiente CLASSPATH:

```
                                        // Installare classi IBM Toolbox
                                        // per Java sul client.
        URL sourceURL = new URL("http://mySystem.myCompany.com/QIBM/ProdData/HTTP/Public/jt400/");

        if (AS400ToolboxInstaller.install(
                        "ACCESS",
                        "d:\\jt400",
                        sourceURL))

    {
                                        // Se le classi IBM Toolbox per Java sono state installate
```

```
                                        // o aggiornate, individuare cosa va aggiunto alla
                                        // variabile d'ambiente CLASSPATH.
            Vector additions = AS400ToolboxInstaller.getClasspathAdditions();

                                        // Se vanno effettuati aggiornamenti di CLASSPATH
    if (additions.size() > 0)
    {
                                        // ... Elaborare ogni aggiunta di classpath
    }
}

                                        // ... Altrimenti non erano necessari aggiornamenti.
```

Utilizzare il metodo isInstalled() per determinare se le classi IBM Toolbox per Java sono già installate sul client.L'utilizzo del metodo isInstalled() consente di determinare se si desidera completare l'installazione ora o terminarla in un momento più conveniente.

Il metodo install() installa e aggiorna i file sul client. Un programma Java può richiamare il metodo isUpdateNeeded() per determinare se è necessario un'aggiornamento prima di richiamare install().

Utilizzare il metodo unInstall() per rimuovere le classi IBM Toolbox per Java dal client.Il metodo unInstall rimuove soltanto i file; la variabile di ambiente CLASSPATH non viene modificata. Richiamare il metodo getClasspathRemovals() per determinare cosa può essere rimosso dalla variabile di ambiente CLASSPATH.

Per ulteriori esempi su come installare la classe AS400ToolboxInstaller all'interno di un programma sulla stazione di lavoro client, fare riferimento all'esempio Installazione/Aggiornamento.

AS400ToolboxJarMaker

Mentre il formato file JAR è stato progettato per velocizzare il processo di scaricamento dei file del programma Java, le classi AS400ToolboxJarMaker creano un processo di caricamento ancora più veloce dei file JAR IBM Toolbox per Java grazie alla loro capacità di creare un file JAR più piccolo da uno più grande.

Inoltre, la classe AS400ToolboxJarMaker può decomprimere un file JAR per poter accedere al singolo file di contenuto individuale per utilizzo di base.

Flessibilità di AS400ToolboxJarMaker

Tutte le funzioni di AS400ToolboxJarMaker vengono eseguite con la classe JarMaker e la sottoclasse AS400ToolboxJarMaker:

- Lo strumento generico JarMaker opera su qualsiasi file JAR o Zip; esso divide un file jar o ne riduce la dimensione rimuovendo le classi che non sono utilizzate.
- AS400ToolboxJarMaker personalizza ed estende le funzioni JarMaker per un utilizzo più semplice con i file JAR IBM Toolbox per Java.

A seconda delle proprie necessità è possibile richiamare il metodo AS400ToolboxJarMaker dall'interno del programma Java o dalla riga comandi. Richiamare AS400ToolboxJarMaker dalla riga comandi utilizzando la sintassi che segue:

```
java utilities.JarMaker [options]
```

dove

- options = una o più delle opzioni disponibili

Per una serie completa di opzioni disponibili da eseguire da una richiesta riga comandi, consultare quanto segue:

- Opzioni per la classe di base JarMaker
- Opzioni estese per la sottoclasse AS00ToolboxJarMaker

Utilizzo di AS400ToolboxJarMaker

E' possibile utilizzare AS400ToolboxJarMaker per gestire i file JAR per effettuare quanto segue:
- Decomprimere un file inserito in un file JAR
- Suddividere un file JAR di grandi dimensioni in più piccoli file JAR
- Escludere tutti i file IBM Toolbox per Java che l'applicazione non andrà ad eseguire

Decompressione di un file JAR

Supponiamo che si desideri decomprimere solo un file inserito in un file JAR. AS400ToolboxJarMaker consente di espandere il file in:
- Indirizzario corrente (extract(jarFile))
- Un altro indirizzario (extract(jarFile, outputDirectory))

Ad esempio, con il seguente codice, si estraggono AS400.class e tutte le classi da essa dipendenti da jt400.jar:

```
java utilities.AS400ToolboxJarMaker -source jt400.jar
        -extract outputDir
        -requiredFile com/ibm/as400/access/AS400.class
```

Divisione di un file JAR singolo in più piccoli file JAR

Supponiamo di dover dividere un file JAR di grandi dimensioni in file JAR più piccoli, in base alla dimensione massima del file Jar desiderata. AS400ToolboxJarMaker, di conseguenza, fornisce la funzione split(jarFile, splitSize).

Nel codice che segue, jt400.jar viene diviso in una serie di file JAR più piccoli, nessuno più grande di 300KB:

```
java utilities.AS400ToolboxJarMaker -split 300
```

Rimozione di file non utilizzati da file JAR

Con AS400ToolboxJarMaker, è possibile escludere qualsiasi file IBM Toolbox per Java che non sia necessario all'applicazione selezionando solo il linguaggio dei componenti di IBM Toolbox per Java, e i CCSID necessari per eseguire l'applicazione. AS400ToolboxJarMaker fornisce inoltre l'opzione per includere o escludere i file JavaBean collegati ai componenti selezionati.

Ad esempio, il comando che segue crea un file JAR che contiene solo quelle classi IBM Toolbox per Java necessarie per creare i componenti CommandCall e ProgramCall della funzione IBM Toolbox per Java:

```
java utilities.AS400ToolboxJarMaker -component CommandCall,ProgramCall
```

Inoltre, se non è necessario convertire stringhe di testo tra Unicode e le tabelle di conversione DBCS (double byte character set), è possibile creare un file JAR più piccolo di 400KB byte omettendo le tabelle di conversione superflue con l'opzione -ccsid:

```
java utilities.AS400ToolboxJarMaker -component CommandCall,ProgramCall -ccsid 61952
```

Nota: le classi di conversione non sono incluse nelle classi ProgramCall. Quando si includono le classi ProgramCall, è necessario includere esplicitamente le classi di conversione utilizzate dal programma, utilizzando l'opzione -ccsid.

Componenti supportati da IBM Toolbox per Java:

La seguente tabella elenca gli ID dei componenti che è possibile specificare quando si richiama lo strumento AS400ToolboxJarMaker.

- La colonna Componente elenca il nome comune per il componente.
- La colonna Parola chiave elenca le parole chiave che è necessario specificare quando si utilizza la tag di opzione `-component`.
- La colonna Costante elenca il valore del numero intero che è necessario specificare in setComponents() e getComponents().

Componente	**Parola chiave**	**Costante**
Oggetto server	AS400	AS400ToolboxJarMaker.AS400
Chiamata comando	CommandCall	AS400ToolboxJarMaker.COMMAND_CALL
Lotto di collegamenti	ConnectionPool	AS400ToolboxJarMaker.CONNECTION_POOL
Aree dati	DataArea	AS400ToolboxJarMaker.DATA_AREA
Conversione e descrizione dei dati	DataDescription	AS400ToolboxJarMaker.DATA_DESCRIPTION
Code dati	DataQueue	AS400ToolboxJarMaker.DATA_QUEUE
Certificati digitali	DigitalCertificate	AS400ToolboxJarMaker.DIGITAL_CERTIFICATE
FTP	FTP	AS400ToolboxJarMaker.FTP
IFS	IntegratedFileSystem	AS400ToolboxJarMaker.INTEGRATED_FILE_SYSTEM
JAAS	JAAS	AS400ToolboxJarMaker.JAAS
Chiamata dell'applicazione Java	JavaApplicationCall	AS400ToolboxJarMaker.JAVA_APPLICATION_CALL
JDBC	JDBC	AS400ToolboxJarMaker.JDBC
Lavori e code lavori	Job	AS400ToolboxJarMaker.JOB
Messaggi e code messaggi	Message	AS400ToolboxJarMaker.MESSAGE
Tipi di dati numerici	NumericDataTypes	AS400ToolboxJarMaker.NUMERIC_DATA_TYPES
NetServer	NetServer	AS400ToolboxJarMaker.NETSERVER
Stampante di rete	Stampa	AS400ToolboxJarMaker.PRINT
Chiamata programma	ProgramCall	AS400ToolboxJarMaker.PROGRAM_CALL
Accesso al livello record	RecordLevelAccess	AS400ToolboxJarMaker.RECORD_LEVEL_ACCESS
Server sicuro	SecureAS400	AS400ToolboxJarMaker.SECURE_AS400
Chiamata al programma del servizio	ServiceProgramCall	AS400ToolboxJarMaker.SERVICE_PROGRAM_CALL
Stato di sistema	SystemStatus	AS400ToolboxJarMaker.SYSTEM_STATUS
Valori di sistema	SystemValue	AS400ToolboxJarMaker.SYSTEM_VALUE
Traccia e registrazione	Traccia	AS400ToolboxJarMaker.TRACE
Utenti e gruppi	User	AS400ToolboxJarMaker.USER

Componente	Parola chiave	Costante
Aree dell'utente	UserSpace	AS400ToolboxJarMaker.USER_SPACE
Oggetto del server visuale	AS400Visual	AS400ToolboxJarMaker.AS400_VISUAL
Chiamata dei comandi visuali	CommandCallVisual	AS400ToolboxJarMaker.COMMAND_CALL_VISUAL
Code dati visuali	DataQueueVisual	AS400ToolboxJarMaker.DATA_QUEUE_VISUAL
IFS visuale	IntegratedFileSystemVisual	AS400ToolboxJarMaker.INTEGRATED_FILE_SYSTEM_VISUAL
Chiamata dell'applicazione Java visuale	JavaApplicationCallVisual	AS400ToolboxJarMaker.JAVA_APPLICATION_CALL_VISUAL
JDBC visuale	JDBCVisual	AS400ToolboxJarMaker.JDBC_VISUAL
Lavori e code lavoro visuali	JobVisual	AS400ToolboxJarMaker.JOB_VISUAL
Messaggi e code messaggi visuali	MessageVisual	AS400ToolboxJarMaker.MESSAGE_VISUAL
Stampante di rete visuale	PrintVisual	AS400ToolboxJarMaker.PRINT_VISUAL
Chiamata del programma visuale	ProgramCallVisual	AS400ToolboxJarMaker.PROGRAM_CALL_VISUAL
Accesso al livello record visuale	RecordLevelAccessVisual	AS400ToolboxJarMaker.RECORD_LEVEL_ACCESS_VISUAL
Utenti e gruppi visuali	UserVisual	AS400ToolboxJarMaker.USER_VISUAL

CCSID e valori di codifica supportati da IBM Toolbox per Java:

L'IBM Toolbox per Java viene fornito con una serie di tabelle di conversione, denominate in base al CCSID. Tali tabelle vengono utilizzate internamente dalle classi IBM Toolbox per Java (come CharConverter) quando si convertono dati che vengono spostati da o a un server iSeries. Ad esempio, la tabella di conversione CCSID 1027 si trova nel file com/ibm/as400/access/ConvTable1027.class. Le tabelle di conversione per i CCSID che seguono sono incluse nel file jar di IBM Toolbox per Java; altre codifiche sono supportate utilizzando JDK. Il server centrale sul server non viene più utilizzato per scaricare le tabelle durante il tempo di esecuzione. Ogni CCSID specificato, per il quale non si trova una tabella di conversione o una codifica JDK darà origine a un'eccezione. Alcune di queste tabelle possono essere ridondanti per le tabelle incluse in JDK. IBM Toolbox per Java attualmente supporta 122 diversi CCSID OS/400.

Per ulteriori informazioni relative ai CCSID, incluso un elenco completo di CCSID riconosciuti dai server iSeries, consultare Globalizzazione.

CCSID supportati in IBM Toolbox per Java

CCSID	Formato	Descrizione
37	EBCDIC single-byte	Stati Uniti e altri
273	EBCDIC single-byte	Austria, Germania
277	EBCDIC single-byte	Danimarca, Norvegia
278	EBCDIC single-byte	Finlandia, Svezia
280	EBCDIC single-byte	Italia

CCSID	Formato	Descrizione
284	EBCDIC single-byte	Spagna, America Latina
285	EBCDIC single-byte	Regno Unito
290	EBCDIC single-byte	Katakana giapponese (solo single-byte)
297	EBCDIC single-byte	Francia
300	EBCDIC double-byte	Giapponese grafico (sottoserie di 16684)
367	ASCII/ISO/Windows	ASCII (ANSI X3.4 standard)
420	EBCDIC single-byte (bidirezionale)	EBCDIC ST4 Arabo
423	EBCDIC single-byte	Greco (per compatibilità; consultare 875)
424	EBCDIC single-byte (bidirezionale)	EBCDIC ST4 Ebraico
437	ASCII/ISO/Windows	ASCII (dati PC USA)
500	EBCDIC single-byte	Latino-1 (MNCS)
720	ASCII/ISO/Windows	Arabo (MS-DOS)
737	ASCII/ISO/Windows	Greco (MS-DOS)
775	ASCII/ISO/Windows	Baltico (MS-DOS)
813	ASCII/ISO/Windows	ISO 8859-7 (Greco/Latino)
819	ASCII/ISO/Windows	ISO 8859-1 (Latino-1)
833	EBCDIC single-byte	Coreano (solo single-byte)
834	EBCDIC double-byte	Coreano grafico (sottoserie di 4930)
835	EBCDIC double-byte	Cinese grafico tradizionale
836	EBCDIC single-byte	Cinese semplificato (solo single-byte)
837	EBCDIC double-byte	Cinese grafico semplificato
838	EBCDIC single-byte	Tailandese
850	ASCII/ISO/Windows	Latino-1
851	ASCII/ISO/Windows	Greco
852	ASCII/ISO/Windows	Latino-2
855	ASCII/ISO/Windows	Cirillico
857	ASCII/ISO/Windows	Turco
860	ASCII/ISO/Windows	Portoghese
861	ASCII/ISO/Windows	Islanda
862	ASCII/ISO/Windows (bidirezionale)	ASCII ST4 ebraico
863	ASCII/ISO/Windows	Canada
864	ASCII/ISO/Windows (bidirezionale)	ASCII ST5 arabo
865	ASCII/ISO/Windows	Danimarca/Norvegia
866	ASCII/ISO/Windows	Cirillico/Russo
869	ASCII/ISO/Windows	Greco
870	EBCDIC single-byte	Latino-2
871	EBCDIC single-byte	Islanda
874	ASCII/ISO/Windows	Tailandese (sottoserie di 9066)
875	EBCDIC single-byte	Greco

CCSID	Formato	Descrizione
878	ASCII/ISO/Windows	Russo
880	EBCDIC single-byte	Multilingue cirillico (per compatibilità; consultare 1025)
912	ASCII/ISO/Windows	ISO 8859-2 (Latino-2)
914	ASCII/ISO/Windows	ISO 8859-4 (Latino-4)
915	ASCII/ISO/Windows	ISO 8859-5 (Cirillico 8-bit)
916	ASCII/ISO/Windows (bidirezionale)	ISO 8859-8 (Ebraico) ST5
920	ASCII/ISO/Windows	ISO 8859-9 (Latino-5)
921	ASCII/ISO/Windows	ISO 8859-13 (Baltico 8-bit)
922	ASCII/ISO/Windows	ISO-8 Estonia
923	ASCII/ISO/Windows	ISO 8859-15 (Latino-9)
930	EBCDIC mixed-byte	Giapponese (sottoserie di 5026)
933	EBCDIC mixed-byte	Coreano (sottoserie di 1364)
935	EBCDIC mixed-byte	Cinese semplificato (sottoserie di 1388)
937	EBCDIC mixed-byte	Cinese tradizionale
939	EBCDIC mixed-byte	Giapponese (sottoserie di 5035)
1025	EBCDIC single-byte	Cirillico
1026	EBCDIC single-byte	Turco
1027	EBCDIC single-byte	Giapponese Latino (solo single-byte)
1046	ASCII/ISO/Windows (bidirezionale)	Windows Arabo ST5
1089	ASCII/ISO/Windows (bidirezionale)	ISO 8859-6 6 (Arabo) ST5
1112	EBCDIC single-byte	Multilingue baltico
1122	EBCDIC single-byte	Estone
1123	EBCDIC single-byte	Ucraino
1125	ASCII/ISO/Windows	Ucraino
1129	ASCII/ISO/Windows	Vietnamita
1130	EBCDIC single-byte	Vietnamita
1131	ASCII/ISO/Windows	Bielorusso
1132	EBCDIC single-byte	Laotiano
1140	EBCDIC single-byte	Stati Uniti e altri (supporto europeo)
1141	EBCDIC single-byte	Austria, Germania (supporto europeo)
1142	EBCDIC single-byte	Danimarca, Norvegia (supporto europeo)
1143	EBCDIC single-byte	Finlandia, Svezia (sopporto europeo)
1144	EBCDIC single-byte	Italia (supporto europeo)
1145	EBCDIC single-byte	Spagna, America Latina (supporto europeo)
1146	EBCDIC single-byte	Regno Unito (supporto europeo)
1147	EBCDIC single-byte	Francia (supporto europeo)
1148	EBCDIC single-byte	Latino-1 (MNCS) (supporto europeo)

CCSID	Formato	Descrizione
1149	EBCDIC single-byte	Islanda (supporto europeo)
1200	Unicode	Unicode UCS-2 (little-endian)
1250	ASCII/ISO/Windows	Windows Latino-2
1251	ASCII/ISO/Windows	Windows Cirillico
1252	ASCII/ISO/Windows	Windows Latino-1
1253	ASCII/ISO/Windows	Windows Greco
1254	ASCII/ISO/Windows	Windows Turco
1255	ASCII/ISO/Windows (bidirezionale)	Windows ebraico ST5
1256	ASCII/ISO/Windows (bidirezionale)	Windows Arabo ST5
1257	ASCII/ISO/Windows	Windows Baltico
1258	ASCII/ISO/Windows	Windows vietnamita
1364	EBCDIC mixed-byte	Giapponese
1388	EBCDIC mixed-byte	Cinese semplificato
1399	EBCDIC mixed-byte	Giapponese (in V4R5 e versioni successive)
4396	EBCDIC double-byte	Giapponese (sottoserie di 300)
4930	EBCDIC double-byte	Coreano
4931	EBCDIC double-byte	Cinese tradizionale (sottoserie di 835)
4933	EBCDIC double-byte	Cinese grafico GBK semplificato
4948	ASCII/ISO/Windows	Latino-2 (sottoserie di 852)
4951	ASCII/ISO/Windows	Cirillico (sottoserie di 855)
5026	EBCDIC mixed-byte	Giapponese
5035	EBCDIC mixed-byte	Giapponese
5123	EBCDIC single-byte	Giapponese (solo single-byte, supporto europeo)
5351	ASCII/ISO/Windows (bidirezionale)	Windows ebraico (supporto europeo) ST5
8492	EBCDIC double-byte	Giapponese (sottoserie di 300)
8612	EBCDIC single-byte	EBCDIC ST5 Arabo
9026	EBCDIC double-byte	Coreano (sottoserie di 834)
9029	EBCDIC double-byte	Cinese semplificato (sottoserie di 4933)
9066	ASCII/ISO/Windows	Tailandese (SBCS esteso)
12588	EBCDIC double-byte	Giapponese (sottoserie di 300)
13122	EBCDIC double-byte	Coreano (sottoserie di 834)
16684	EBCDIC double-byte	Giapponese (disponibile in V4R5)
17218	EBCDIC double-byte	Coreano (sottoserie di 834)
12708	EBCDIC single-byte	EBCDIC ST7 Arabo
13488	Unicode	Unicode UCS-2 (big-endian)
28709	EBCDIC single-byte	Cinese tradizionale (solo single-byte)
61952	Unicode	Unicode iSeries (utilizzato principalmente nell'IFS)

CCSID	Formato	Descrizione
62211	EBCDIC single-byte	EBCDIC ST5 Ebraico
62224	EBCDIC single-byte	EBCDIC ST6 Arabo
62235	EBCDIC single-byte	EBCDIC ST6 Ebraico
62245	EBCDIC single-byte	EBCDIC ST10 Ebraico

Classe CommandHelpRetriever

La classe CommandHelpRetriever richiama il testo di aiuto per i comandi CL (control language) OS/400 e genera tale testo in formato HTML o UIM (User Interface Manager). E' possibile eseguire CommandHelpRetriever da una riga comandi o incorporare la funzione nel proprio programma Java.

Per utilizzare CommandHelpRetriever, sul server deve essere in esecuzione OS/400 V5R1 o successivi e deve essere presente un programma di analisi XML ed un processore XSL nella variabile di ambiente CLASSPATH. Per ulteriori informazioni, consultare la seguente pagina:

"Programma di analisi XML e processore XSLT" a pagina 417

Inoltre, il comando CL GENCMDDOC (Creazione documentazione comando) utilizza il comando CommandHelpRetriever. Quindi è possibile semplicemente utilizzare il comando GENCMDDOC per trarre vantaggio dalla funzionalità offerta dalla classe CommandHelpRetriever. Per ulteriori informazioni, consultare la seguente pagina:

GENCMDDOC (Creazione documentazione comando)

Esecuzione di CommandHelpRetriever da una riga comandi

E' possibile eseguire la classe CommandHelpRetriever come un programma autonomo della riga comandi. Per eseguire CommandHelpRetriever da una riga comandi, è necessario passare come minimo i seguenti parametri:

- La libreria sul server iSeries che contiene il comando CL. I comandi di sistema si trovano nella libreria QSYS.
- Il comando CL.

E' possibile anche passare a CommandHelpRetriever parametri facoltativi che includono il server iSeries, l'ID utente, la parola d'ordine e l'ubicazione per il file creato.

Per ulteriori informazioni, consultare la documentazione di riferimento Javadoc per CommandHelpRetriever.

Esempio: utilizzo di CommandHelpRetriever da una riga comandi

Il seguente esempio genera un file HTML denominato CRTLIB.html nell'indirizzario corrente.

Nota: il comando di esempio viene posto su due righe solo a scopo di visualizzazione. Immettere il comando su un'unica riga.

```
java com.ibm.as400.util.CommandHelpRetriever -library QSYS -command CRTLIB
     -system MySystem -userid MyUserID -password MyPassword
```

Come incorporare la classe CommandHelpRetriever nel programma

E' possibile anche utilizzare la classe CommandHelpRetriever nell'applicazione Java per visualizzare la documentazione di aiuto per i comandi CL specificati. Dopo aver creato un oggetto

CommandHelpRetriever, è possibile utilizzare i metodi generateHTML e generateUIM per creare la documentazione di aiuto nell'uno o nell'altro formato.

Quando si utilizza generateHTML(), è possibile visualizzare l'HTML creato nel gruppo di pannelli per il comando oppure si può specificare un gruppo di pannelli differente.

Il seguente esempio crea un oggetto CommandHelpRetriever e genera oggetti Stringa che rappresentano la documentazione di aiuto HTML e UIM per il comando CRTLIB.

```
CommandHelpRetriever helpGenerator = new CommandHelpRetriever();
AS400 system = new AS400("MySystem", "MyUserID", "MyPassword");
Command crtlibCommand = new Command(system, "/QSYS.LIB/CRTLIB.CMD");
String html = helpGenerator.generateHTML(crtlibCommand);
String uim = helpGenerator.generateUIM(crtlibCommand);
```

Documentazione di riferimento Javadoc

Per ulteriori informazioni sulla classe CommandHelpRetriever, consultare la seguente documentazione di riferimento Javadoc:

CommandHelpRetriever

Classe CommandPrompter

La classe CommandPrompter richiede il parametro su un dato comando. Il CommandPrompter fornisce una funzionalità che è simile alla richiesta comandi CL di iSeries (premendo F4) e lo stesso di Richiesta comandi di Management Central.

Per utilizzare il CommandPrompter, è necessario che il server esegua OS/400 V4R4 o versioni successive. Per ulteriori informazioni, consultare la pagina iSeries Navigator Information APAR e consultare Required Fixes for Graphical Command Prompter Support.

L'utilizzo di CommandPrompter richiede anche che i seguenti file jar siano in CLASSPATH:

- jt400.jar
- jui400.jar
- util400.jar
- jhall.jar

E' inoltre necessario disporre di un programma di analisi XML nel CLASSPATH. Per ulteriori informazioni sulle modalità di utilizzo di un programma di analisi XML appropriato, consultare la pagina seguente:

"Programma di analisi XML e processore XSLT" a pagina 417

Tutti i file jar, ad eccezione del file jhall.jar, sono inclusi in IBM Toolbox per Java. Per ulteriori informazioni sui file jar di IBM Toolbox per Java, consultare File jar. Per ulteriori informazioni su come scaricare il file jhall.jar, consultare il sito Web Sun JavaHelp(TM) .

Per creare un oggetto CommandPrompter, passare i parametri per la frame principale che lancia la richiesta, l'oggetto AS400 sul quale il comando verrà richiesto e la stringa di comando. La stringa di comando può essere un nome di comando, una stringa di comando completa o un nome di comando parziale, come `crt*`.

Il pannello CommandPrompter è una finestra di dialogo modale che l'utente deve chiudere prima di tornare alla frame principale. CommandPrompter gestisce tutti gli errori incontrati durante la richiesta. Per visualizzare un esempio di programmazione che mostri un modo per utilizzare CommandPrompter, consultare la pagina seguente:

"Esempio: utilizzo di CommandPrompter" a pagina 715

RunJavaApplication

Le classi RunJavaApplication e VRunJavaApplication sono programmi di utilità per l'esecuzione di programmi Java sulla JVM iSeries.A differenza delle classi JavaApplicationCall e VJavaApplicationCall richiamate dal programma Java, RunJavaApplication e VRunJavaApplication sono programmi completi.

La classe RunJavaApplication è un programma di utilità a riga comandi. Consente di impostare l'ambiente (CLASSPATH e proprietà, ad esempio) per il programma Java. Specificare il nome del programma Java e i parametri, quindi avviare il programma. Una volta avviato, è possibile inviare immissioni al programma Java che vengono ricevute tramite immissione standard. Il programma Java crea emissione per emissione standard e errori standard.

Il programma di utilità VRunJavaApplication ha le stesse capacità. La differenza consiste nel fatto che VJavaApplicationCall utilizza una GUI (Graphical User Interface) mentre JavaApplicationCall è un'interfaccia a riga comandi.

JPing

La classe JPing è un programma di utilità della riga comandi che consente di interrogare i server per vedere quali servizi sono in esecuzione e quali porte sono in attività.Per interrogare i server dall'interno di un'applicazione Java, utilizzare la classe AS400JPing.

Consultare javadoc JPing per ulteriori informazioni sull'utilizzo di JPing dall'interno dell'applicazione Java.

Richiamare JPing dalla riga comandi utilizzando la seguente sintassi:

```
java utilities.JPing System [options]
```

dove:

- System = il server iSeries che si desidera interrogare
- [options] = una o più opzioni disponibili

Opzioni

E' possibile utilizzare una o più delle seguenti opzioni. Per le opzioni fornite di abbreviazioni, queste sono elencate tra parentesi.

-help (-h or -?)
Visualizza il testo aiuto.

-service*OS/400_Service***(-s***OS/400_Service***)**
Specifica un determinato servizio su cui eseguire il ping. L'operazione predefinita è l'esecuzione del ping di tutti i servizi. E' possibile utilizzare questa opzione per specificare uno dei seguenti servizi: as-file, as-netprt, as-rmtcmd, as-dtaq, as-database, as-ddm, as-central e as-signon.

-ssl Specifica se effettuare o meno il ping delle porte ssl. L'operazione predefinita è non eseguire il ping delle porte ssl.

-timeout (-t)
Specifica il periodo di supero tempo in millisecondi. L'impostazione predefinita è di 20000 o 20 secondi.

Esempio: utilizzo di JPing da una riga comandi

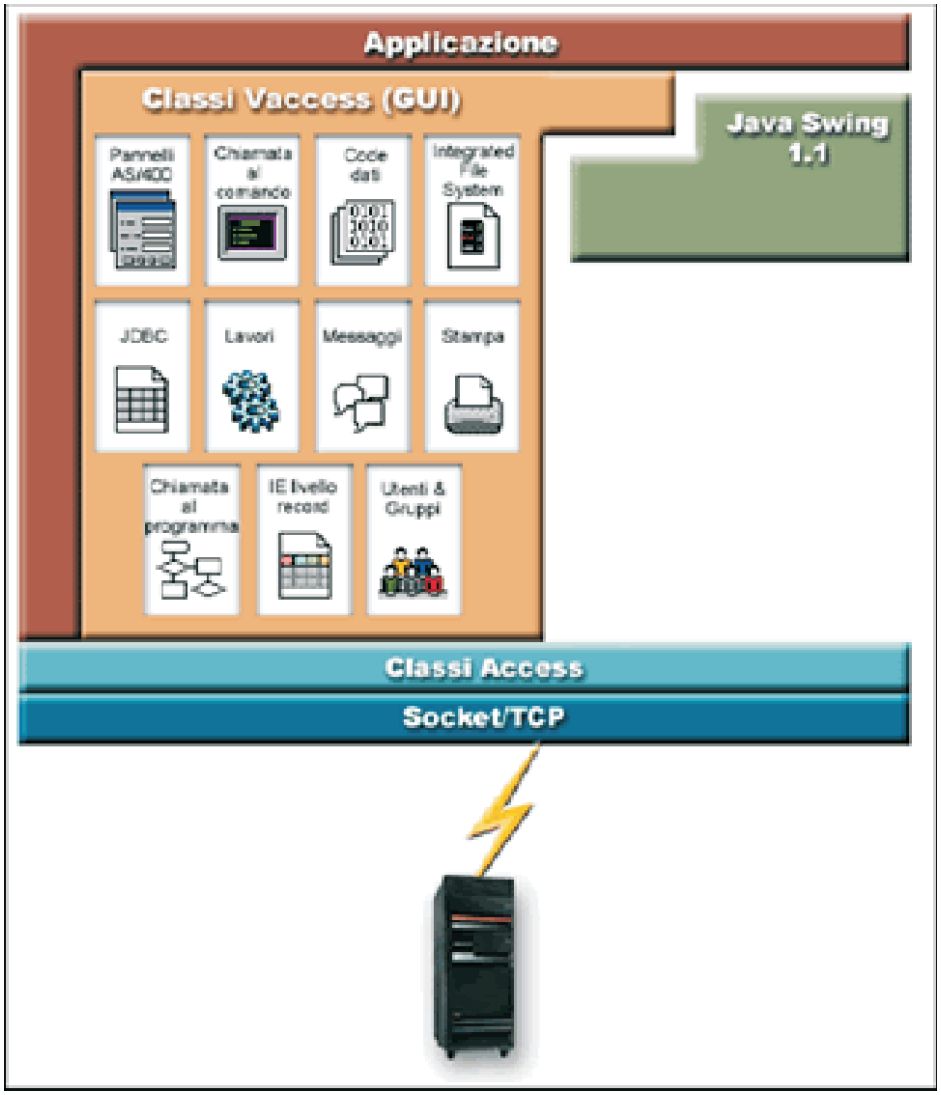
Ad esempio, utilizzare il seguente comando per effettuare il ping del servizio as-dtaq, includendo le porte ssl, con un supero tempo di 5 secondi:

```
java utilities.JPing myServer -s as-dtaq -ssl -t 5000
```

Classi Vaccess

IBM Toolbox per Java fornisce le classi GUI (Graphical User Interface) nel pacchetto vaccess per richiamare, visualizzare e in alcuni casi gestire i dati del server. Tali classi utilizzano la struttura Java Swing 1.1. La figura 1 visualizza la relazione tra queste classi:

Figura 1: classi Vaccess



"Descrizione lunga della Figura 1: classi Vaccess (rzahh508.gif)" a pagina 256

Classi Vaccess

IBM Toolbox per Java fornisce un insieme di classi GUI (graphical user interface) nel pacchetto vaccess.Queste classi utilizzano le classi di accesso per richiamare dati e presentare i dati all'utente.

I programmi Java che utilizzano le classi vaccess di IBM Toolbox per Java richiedono il pacchetto Swing, fornito con Java 2 Platform, Standard Edition (J2SE). Per ulteriori informazioni su Swing, consultare il sito web di Sun Java Foundation Classes .

Per ulteriori informazioni sulla relazione tra le classi GUI di IBM Toolbox per Java, le classi Access e Java Swing, consultare Diagramma di classi Vaccess.

Utilizzare le classi pannelli AS400 per visualizzare i dati iSeries.

Le API sono disponibili per accedere alle seguenti risorse iSeries e ai relativi strumenti:
- Chiamata al comando
- Code dati
- Eventi errore*
- Integrated file system
- JavaApplicationCall
- JDBC
- Lavori*
- Messaggi*
- Autorizzazione
- Stampa* incluso il programma di visualizzazione file di spool
- ProgramCall e ProgramParameter
- Accesso al livello del record
- Elenchi risorse
- Stato del sistema
- Valori di sistema
- Utenti e gruppi

Nota: i pannelli AS400 vengono utilizzati con altre classi Vaccess (consultare le voci precedenti contrassegnate con un asterisco) per presentare e consentire la gestione delle risorse iSeries.

Ogni volta che si esegue una programmazione con i componenti GUI di IBM Toolbox per Java, utilizzare le classi di Eventi errore per notificare all'utente gli eventi errore e per gestirli.

Per ulteriori informazioni relative all'accesso ai dati iSeries, consultare Classi Access.

Descrizione lunga della Figura 1: classi Vaccess (rzahh508.gif):

In IBM Toolbox per Java: diagramma pacchetto Vaccess

Questa figura mostra la relazione tra le classi nel pacchetto vaccess, nel pacchetto access e nelle classi Java Swing.

Descrizione

La figura è composta da quanto segue:
- L'immagine nella parte più alta in realtà è un bordo marrone spesso, come a formare i lati sinistro e superiore del rettangolo, etichettato 'Applicazione', che rappresenta un'applicazione Java. Il rettangolo delineato dall'applicazione Java contiene le seguenti immagini di forma irregolare che si adattano perfettamente l'una con l'altra come se fossero pezzi di un puzzle:
- Un poligono rossiccio che rappresenta le classi (GUI) vaccess di IBM Toolbox per Java. Questa figura contiene immagini più piccole che rappresentano le funzioni contenute nelle classi vaccess.

- Un poligono verde che rappresenta le classi Java Swing
- Sotto queste immagini vi è una barra blu chiaro etichettata Classi Access, che rappresenta classi nel pacchetto access di IBM Toolbox per Java.
- Sotto questa barra di colore blu chiaro vi è una barra di colore blu scuro identica, etichettata 'Socket/TCP.'
- L'immagine in basso è la rappresentazione di un server iSeries.
- Una freccia a forma di fulmine, che rappresenta un collegamento socket dall'applicazione Java al server, si estende verso il basso da Socket/TCP (la barra blu scuro) e si collega al server (l'immagine di un server iSeries).

Un'applicazione Java (l'area delineata dal bordo marrone spesso) contiene classi vaccess (il poligono rossiccio) e classi Java Swing (il poligono verde). Le classi vaccess abilitano l'applicazione Java ad accedere ai seguenti dati e funzioni sul server (le immagini piccole nel poligono rossiccio):

Pannelli AS400, chiamata al comando, code dati, IFS (integrated file system), JDBC, lavori, messaggi, stampa, chiamata al programma, immissione ed emissione al livello record e utenti e gruppi

L'applicazione Java utilizza le classi access di IBM Toolbox per Java (la barra blu chiaro) per creare una o più collegamenti socket (la barra blu scuro). I collegamenti socket abilitano l'applicazione Java a comunicare (la freccia a forma di fulmine) con il server (l'immagine in basso del server iSeries).

AS400Panes

AS400Panes sono componenti nel pacchetto vaccess che presentano e consentono la manipolazione di una o più risorse del server in una GUI. La funzionalità di ogni risorsa del server varia a seconda del tipo di risorsa.

Tutti i pannelli estendono la classe Java Component. Di conseguenza, possono essere aggiunti a ogni AWT Frame, finestra o contenitore.

Sono disponibili i seguenti AS400Panes:
- AS400DetailsPane presenta un elenco di risorse del server in una tabella dove ogni riga visualizza dettagli vari su ogni singola risorsa. La tabella consente la selezione di una o più risorse.
- AS400ExplorerPane unisce un AS400TreePane e un AS400DetailsPane in modo che la risorsa selezionata nell'albero viene presentata nei dettagli.
- AS400JDBCDataSourcePane presenta i valori di proprietà di un oggetto AS400JDBCDataSource.
- AS400ListPane presenta un elenco delle risorse del server e consente la selezione di una o più risorse.
- AS400TreePane presenta una gerarchia ad albero delle risorse del server e consente la selezione di una o più risorse.

Risorse server

Le risorse del server sono rappresentate nella GUI con una icona e un testo. Le risorse del server sono definite con relazioni gerarchiche e una di esse può avere una risorsa principale e zero o più risorse secondarie. Si tratta di relazioni predefinite utilizzate per specificare quali risorse vengono visualizzate in un AS400Pane. Ad esempio, VJobList è la risorsa principale di zero o più VJobs e tale relazione gerarchica è rappresentata graficamente in un AS400Pane.

IBM Toolbox per Java fornisce l'accesso alle seguenti risorse del server:
- VIFSDirectory rappresenta un indirizzario nell'IFS.
- VJob e VJobList rappresentano un lavoro o un elenco di lavori
- VMessageList e VMessageQueue rappresentano un elenco di messaggi restituiti da una CommandCall o da una ProgramCall o una coda di messaggi

- VPrinter, VPrinters e VPrinterOutput rappresentano una stampante, un elenco di stampanti o un elenco di file di spool
- VUserList rappresenta un elenco di utenti

Tutte le risorse sono implementazioni dell'interfaccia VNode.

Impostazione della root

Per specificare quali risorse del server sono presentate in un AS400Pane, impostare la root utilizzando il programma di creazione o il metodo setRoot(). La root definisce l'oggetto del primo livello e viene utilizzato a seconda dei casi in base al pannello:

- AS400ListPane presenta tutti i valori secondari della root nel proprio elenco.
- AS400DetailsPane presenta tutti i valori secondari della root nella propria tabella
- AS400TreePane utilizza la root come root del proprio albero
- AS400ExplorerPane utilizza la root come root del proprio albero

Sono possibili tutte le combinazioni di pannelli e root.

Il seguente esempio crea un AS400DetailsPane per presentare l'elenco di utenti definito sul sistema:

```
                                        // Creare la risorsa server
                                        // che rappresenta un elenco di utenti.
                    // Supponiamo che "system" sia un oggetto AS400
                    // creato ed inizializzato
                    // da qualche altra parte.
          VUserList userList = new VUserList (system);

                                        // Creare l'oggetto AS400DetailsPane
                                        // ed impostare la relativa root in modo che sia
                                        // l'elenco utenti.
  AS400DetailsPane detailsPane = new AS400DetailsPane ();
  detailsPane.setRoot (userList);

                    // Aggiungere il pannello dettagli ad una frame.
                                        // Supponiamo che "frame" sia un JFrame
                    // creato da qualche altra parte.
frame.getContentPane().add(detailsPane);
```

Caricamento del contenuto

Quando vengono creati gli oggetti AS400Pane e gli oggetti della risorsa del server, vengono inizializzati su uno stato predefinito. Le informazioni rilevanti che compongono il contenuto del pannello non vengono caricate al momento della creazione.

Per caricare il contenuto, è necessario che l'applicazione richiami esplicitamente il metodo load(). Nella maggior parte dei casi, questo inizializza la comunicazione al server per raccogliere informazioni rilevanti. Poiché a volte è necessario qualche secondo per raccogliere le informazioni, l'applicazione può controllare esattamente quando ciò avviene. Ad esempio, è possibile:

- Caricare il contenuto prima di aggiungere il pannello a una frame. La frame non viene visualizzata prima del caricamento completo delle informazioni.
- Caricare il contenuto dopo aver aggiunto il pannello a una frame e visualizzato tale frame. La frame viene visualizzata ma non contiene molte informazioni. Viene visualizzata una clessidra (cursore di attesa) e il campo informazioni viene riempito durante il suo caricamento.

Il seguente esempio carica il contenuto di un pannello dettagli prima di aggiungerlo ad una frame:

```
                                        // Caricare il contenuto del pannello
                                        // dettagli. Supponiamo che il detailsPane
                  // sia stato creato ed inizializzato
```

```
                    // da qualche altra parte.
          detailsPane.load ();

                    // Aggiungere il pannello dettagli ad una frame.
                                        // Supponiamo che "frame" sia un JFrame
                    // creato da qualche altra parte.
frame.getContentPane().add(detailsPane);
```

Pannelli delle operazioni e delle proprietà

Nel tempo di esecuzione, l'utente può selezionare un menu a comparsa su ogni risorsa del server. Il menu a comparsa presenta un elenco di operazioni rilevanti, disponibili per la risorsa. Quando l'utente seleziona un'operazione dal menu a comparsa, quell'operazione viene eseguita. Ogni risorsa ha diverse operazioni definite.

In alcuni casi, il menu a comparsa presenta anche una voce che consente all'utente di visualizzare un pannello proprietà. Un pannello proprietà illustra diversi dettagli sulla risorsa e può consentire all'utente di modificare quei dettagli.

L'applicazione può controllare se i pannelli operazioni e proprietà sono disponibili, utilizzando il metodo setAllowActions() sul pannello.

Modelli

I AS400Panes vengono implementati utilizzando il paradigma programma di controllo vista modello, in cui i dati e l'interfaccia dell'utente sono separati in classi diverse. I AS400Panes integrano i modelli IBM Toolbox per Java con i componenti GUI Java. I modelli gestiscono le risorse del server, i componenti vaccess li visualizzano graficamente e gestiscono l'interazione dell'utente.

I AS400Panes forniscono sufficiente funzionalità per la maggior parte dei requisiti. Tuttavia, se un'applicazione richiede un ulteriore controllo del componente JFC, potrà accedere direttamente ad un modello del server e fornire un'integrazione personalizzata con un diverso componente vaccess.

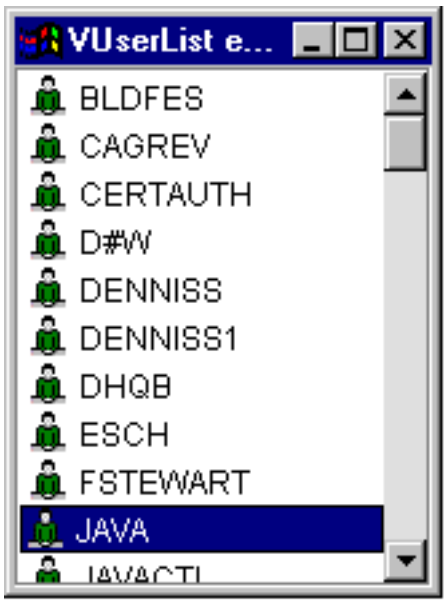
I seguenti modelli sono disponibili:

- AS400ListModel implementa l'interfaccia ListModel JFC come elenco delle risorse del server.Può essere utilizzato con un oggetto JList JFC.
- AS400DetailsModel implementa l'interfaccia TableModel JFC come tabella delle risorse del server in cui ogni riga contiene diversi dettagli su una singola risorsa.Può essere utilizzato con un oggetto JTable JFC.
- AS400TreeModel implementa l'interfaccia TreeModel JFC come gerarchia dell'albero delle risorse del server.Può essere utilizzato con un oggetto JTree JFC.

Esempi

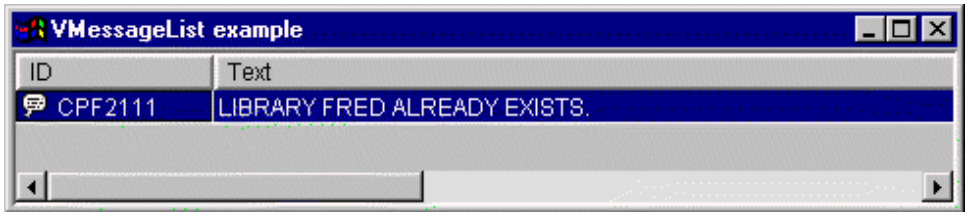
- Presenta un elenco di utenti sul sistema utilizzando un AS400ListPane con un oggetto VUserList. La figura 1 illustra il prodotto finito:

  Figura 1: utilizzo di AS400ListPane con un oggetto VUserList



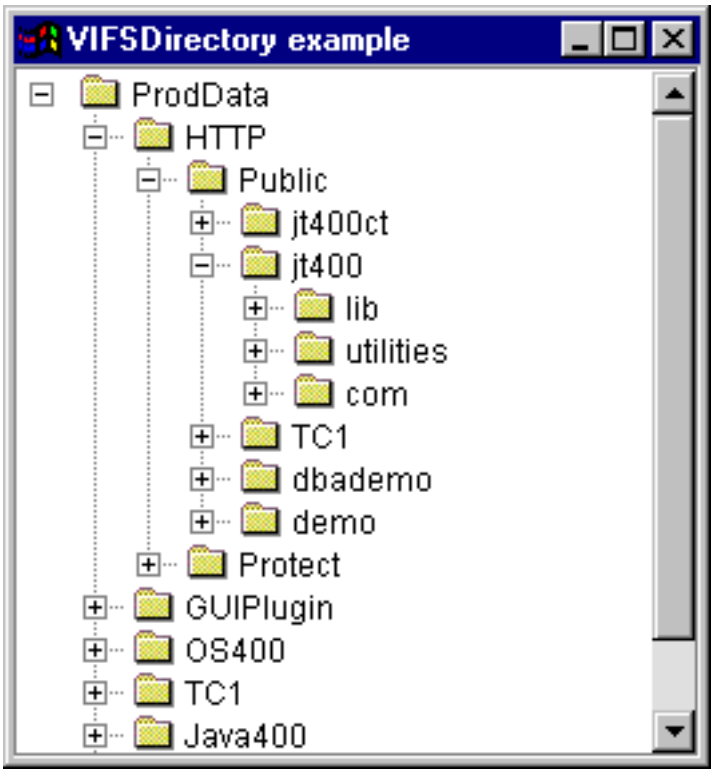
- Presentare l'elenco dei messaggi creati da una chiamata al comando utilizzando un AS400DetailsPane con un oggetto VMessageList.La figura 2 illustra il prodotto finito:

  Figura 2: utilizzo di AS400DetailsPane con un oggetto VMessageList



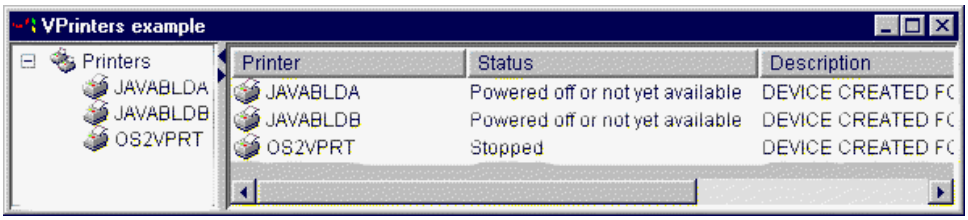
- Presentare una gerarchia dell'indirizzario IFS utilizzando un AS400TreePane con un oggetto VIFSDirectory.La figura 3 illustra il prodotto finito:

  Figura 3: utilizzo di AS400TreePane con un oggetto VIFSDirectory



- Presentare risorse di stampa utilizzando un AS400ExplorerPane con un oggetto VPrinters.La figura 4 illustra il prodotto finito:

  Figura 4: utilizzo di AS400ExplorerPane con un oggetto VPrinters



Chiamata comando

I componenti GUI vaccess di chiamata al comando consentono a un programma Java di presentare un pulsante o una voce di menu che richiama un comando server non interattivo.

Un oggetto CommandCallButton rappresenta un pulsante che, quando premuto, richiama un comando server.La classe CommandCallButton estende la classe JButton JFC (Java Foundation Classes) in modo tale che tutti i pulsanti abbiano un aspetto e un funzionamento coerente.

In modo simile, un oggetto CommandCallMenuItem rappresenta una voce di menu che, quando selezionato, richiama un comando server.La classe CommandCallMenuItem estende la classe JFC JMenuItem in modo tale che tutte le voci di menu abbiano un aspetto e un funzionamento coerente.

Per utilizzare un componente GUI (Graphical User Interface) di chiamata al comandi, impostare le proprietà sistema e comando. Tali proprietà possono essere impostate utilizzando un programma di creazione o tramite i metodi setSystem() e setCommand().

Il seguente esempio crea CommandCallButton. Durante l'esecuzione, quando si preme il pulsante, viene creata una libreria denominata "FRED":

```
                    // Creare l'oggetto CommandCallButton.
                                          // AS400TreePane.
Supponiamo che "system" sia un oggetto
                                          // AS400 creato e inizializzato da qualche
                                          // altra parte.
Il pulsante
                                          // cita "Press Me" e non esiste alcuna
                                          // icona.
    CommandCallButton button = new CommandCallButton ("Press Me", null, system);

                                          // Impostare il comando che verrà eseguito dal pulsante.
    button.setCommand ("CRTLIB FRED");

                    // Aggiungere il pulsante ad una frame. Supponiamo
                    // che "frame" sia un JFrame creato da qualche altra
                    // da qualche altra parte.
       frame.getContentPane ().add (button);
```

Quando viene eseguito un comando server, potrebbe restituire zero o più messaggi del server. Per individuare quando viene eseguito il comando server, aggiungere un ActionCompletedListener al pulsante o alla voce di menu utilizzando il metodo addActionCompletedListener(). All'esecuzione del comando, viene emesso ActionCompletedEvent per tutti i listener. Un listener può utilizzare il metodo getMessageList() per richiamare i messaggi del server generati dal comando.

Questo esempio aggiunge un ActionCompletedListener che elabora tutti i messaggi server che il comando ha generato:

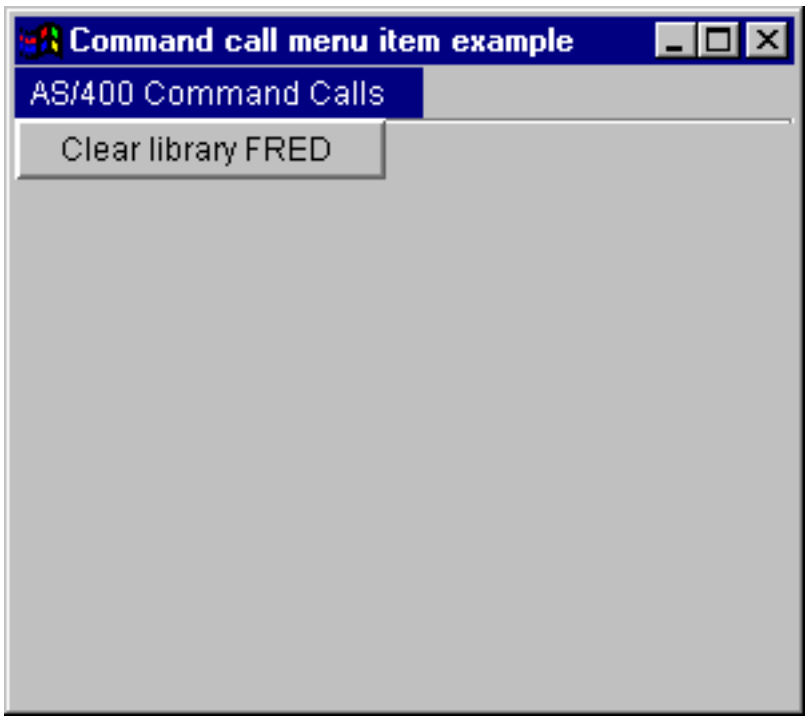
```
                    // Aggiungere un ActionCompletedListener che viene
                                          // implementato utilizzando una classe interna
                                          // anonima. Questo è un modo conveniente per
                                          // specificare  i listener di un evento
                                          // semplice.
               button.addActionCompletedListener (new ActionCompletedListener ()
    {
                       public void actionCompleted (ActionCompletedEvent event)
        {
                                          // Inserire il sorgente dell'evento in un
                                          // CommandCallButton.
                           CommandCallButton sourceButton = (CommandCallButton) event.getSource ();

                    // Richiamare l'elenco dei messaggi server
                    // generati dal comando.
            AS400Message[] messageList = sourceButton.getMessageList ();

                    // ... Elaborare l'elenco dei messaggi.
        }
    });
```

Esempi

Questo esempio mostra come utilizzare un CommandCallMenuItem in un'applicazione.

La Figura 1 visualizza il componente della GUI (Graphical User Interface) CommandCall:

Figura 1: componente GUI CommandCall



Code dati

I componenti grafici della coda dati consentono a un programma Java di utilizzare tutti i componenti testo grafico JFC (Java Foundation Classes) per la lettura o la scrittura in una coda dati server.

Le classi DataQueueDocument e KeyedDataQueueDocument sono implementazioni dell'interfaccia documento JFC.E' possibile utilizzare direttamente queste classi con tutti i componenti testo grafico JFC. Diversi componenti di testo, come i campi a riga singola (JTextField) e le aree testo a più righe (JTextArea), sono disponibili in JFC.

I documenti coda dati associano il contenuto di un componente testo a una coda dati server. (Un componente testo è un componente grafico utilizzato per visualizzare testo modificabile in via opzionale dall'utente.) Il programma Java può leggere e scrivere tra il componente testo e la coda dati in qualsiasi momento. Utilizzare DataQueueDocument per code dati **sequenziali** eKeyedDataQueueDocument per code dati **con chiave**.

Per utilizzare un DataQueueDocument, impostare le proprietà sistema e percorso. E' possibile impostare tali proprietà utilizzando un programma di creazione o tramite i metodi setSystem() e setPath(). L'oggetto DataQueueDocument viene quindi collegato al componente di testo, solitamente utilizzando il programma di creazione del componente testo o il metodo setDocument(). I KeyedDataQueueDocuments funzionano allo stesso modo.

Il seguente esempio crea un DataQueueDocument il cui contenuto è associato ad una coda dati:

```
                                        // Creare l'oggetto DataQueueDocument.
                                        // AS400TreePane.
Supponiamo che "system" sia un oggetto
                                        // AS400 creato e inizializzato da qualche
                                        // altra parte.
     DataQueueDocument dqDocument = new DataQueueDocument (system, "/QSYS.LIB/MYLIB.LIB/MYQUEUE.DTAQ");

                                        // Creare un'area di testo per presentare il
                                        // documento.
          JTextArea textArea = new JTextArea (dqDocument);
```

```
                                        // Aggiungere l'area di testo ad una frame.
                                        // Supponiamo che "frame" sia un JFrame
                    // creato da qualche altra parte.
frame.getContentPane ().add (textArea);
```

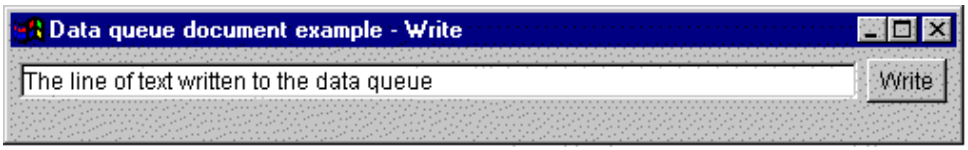
Inizialmente, il contenuto del componente testo è vuoto. Utilizzare read() o peek() per riempire il contenuto con la voce successiva nella coda. Utilizzare write() per scrivere il contenuto del componente testo nella coda dati. Tenere presente che tali documenti funzionano solamente con voci coda dati Stringa.

Esempi

Esempio di utilizzo di un DataQueueDocument in un'applicazione.

La Figura 1 mostra il componente GUI (Graphical User Interface) DataQueueDocument utilizzato in un JTextField. E' stato aggiunto un pulsante per fornire un'interfaccia GUI per l'utente al fine di scrivere il contenuto del campo testo nella coda dati.

Figura 1: componente GUI DataQueueDocument



Eventi di errore

In molti casi, i componenti GUI (Graphical User Interface) di IBM Toolbox per Java emettono eventi errore invece di lanciare delle eccezioni.

Un evento errore è un programma di riavvolgimento ciclico di un'eccezione che viene lanciata da un componente interno.

E' possibile fornire un listener di errore che gestisce tutti gli eventi errore emessi da un particolare componente GUI. Ogni volta che viene inviata un'eccezione, viene richiamato il listener ed è possibile fornire un documentazione appropriata dell'errore. Per impostazione predefinita, non si verifica alcuna operazione quando vengono emessi gli eventi errore.

L'IBM Toolbox per Java fornisce un componente GUI chiamato ErrorDialogAdapter, che consente all'utente di visualizzare automaticamente una finestra di dialogo tutte le volte in cui si verifica un evento errore.

Esempi

I seguenti esempi mostrano come si possono gestire gli errori e definire un semplice listener dell'errore.

Esempio: gestione eventi errore tramite visualizzazione di una finestra di dialogo

Il seguente esempio mostra come gestire gli eventi errore visualizzando una finestra di dialogo:

```
// E' stato effettuato tutto il lavoro di impostazione per il lay out di un componente della
// GUI. Aggiungere ora un ErrorDialogAdapter come listener per il componente.
// Quest'ultimo, segnalerà tutti gli eventi di errore lanciati da tale componente attraverso
// la visualizzazione di una finestra di dialogo.

    ErrorDialogAdapter errorHandler = new ErrorDialogAdapter (parentFrame);
    component.addErrorListener (errorHandler);
```

Esempio: definizione di un listener di errori

E' possibile scrivere un listener errore di personalizzazione per gestire gli errori in un modo differente. Utilizzare l'interfaccia ErrorListener per effettuarlo.

Il seguente esempio mostra come definire un semplice listener di errore che stampa solo gli errori in System.out:

```
      class MyErrorHandler
      implements ErrorListener
{
   // Questo metodo viene richiamato ogni qual volta viene emesso un evento di errore.
               public void errorOccurred(ErrorEvent event)
   {
                        Exception e = event.getException ();
                        System.out.println ("Error: " + e.getMessage ());
   }
}
```

Esempio: gestione degli eventi errore utilizzando un listener di errore

Il seguente esempio mostra come gestire gli eventi errore per un componente GUI utilizzando questo handler personalizzato:

```
MyErrorHandler errorHandler = new MyErrorHandler ();
component.addErrorListener (errorHandler);
```

IFS (Integrated file system)

I componenti della GUI IFS consentono a un programma Java di presentare gli indirizzari e i file nell'IFS (Integrated File System) sul server in una GUI.

I seguenti componenti sono disponibili:

- IFSFileSystemView fornisce un gateway all'IFS iSeries.
- IFSFileDialog presenta una finestra di dialogo che consente all'utente di selezionare un indirizzario e di selezionare un file scorrendo la gerarchia degli indirizzari.
- VIFSDirectory è una risorsa che rappresenta un'indirizzario nell'IFS (Integrated File System) da utilizzare in AS400Panes.
- IFSTextFileDocument rappresenta un file di testo da utilizzare in tutti i componenti di testo grafici JFC (Java Foundation Classes).
- Per utilizzare i componenti della GUI IFS, impostare le proprietà del percorso o dell'indirizzario. Queste proprietà possono essere impostate utilizzando un programma di creazione o tramite i metodi setDirectory() (per IFSFileDialog) o setSystem() e setPath() (per VIFSDirectory e IFSTextFileDocument).

Impostare il percorso su un valore diverso da ″/QSYS.LIB″ poiché questo indirizzario in genre è di grandi dimensioni e scaricarne il contenuto può richiedere molto tempo.

IFSFileSystemView:

IFSFileSystemView fornisce un gateway all'IFS iSeries, da utilizzare nella creazione di oggetti javax.swing.JFileChooser.

JFileChooser è una modalità Java standard per creare finestre di dialogo per esaminare e scegliere i file ed è la sostituzione consigliata per IFSFileDialog.

Esempio: utilizzo di IFSFileSystemView

Il seguente esempio dimostra l'utilizzo di IFSFileSystemView.

```
import com.ibm.as400.access.AS400;
import com.ibm.as400.access.IFSJavaFile;
import com.ibm.as400.vaccess.IFSFileSystemView;
```

```
import javax.swing.JFileChooser;
import java.awt.Frame;

// Gestire l'indirizzario /Dir sul sistema myAS400.
AS400 system = new AS400("myAS400");
IFSJavaFile dir = new IFSJavaFile(system, "/Dir");
JFileChooser chooser = new JFileChooser(dir, new IFSFileSystemView(system));
Frame parent = new Frame();
int returnVal = chooser.showOpenDialog(parent);
if (returnVal == JFileChooser.APPROVE_OPTION) {
   IFSJavaFile chosenFile = (IFSJavaFile)(chooser.getSelectedFile());
   System.out.println("You selected the file named " +
                      chosenFile.getName());
}
```

Finestre di dialogo file:

La classe IFSFileDialog è una finestra di dialogo che consente all'utente di attraversare gli indirizzari dell'IFS (Integrated File System) sul server e selezionare un file.Il programma di chiamata può impostare il testo sui pulsanti nella finestra di dialogo. Inoltre, il programma di chiamata può utilizzare gli oggetti FileFilter, che consentono all'utente di limitare le scelte a determinati file.

Se l'utente seleziona un file nella finestra di dialogo, utilizzare il metodo getFileName() per determinare il nome del file selezionato.Utilizzare il metodo getAbsolutePath() per determinare il nome del percorso completo del file selezionato.

L'esempio seguente imposta una finestra di dialogo file IFS (Integrated File System) con due filtri file:

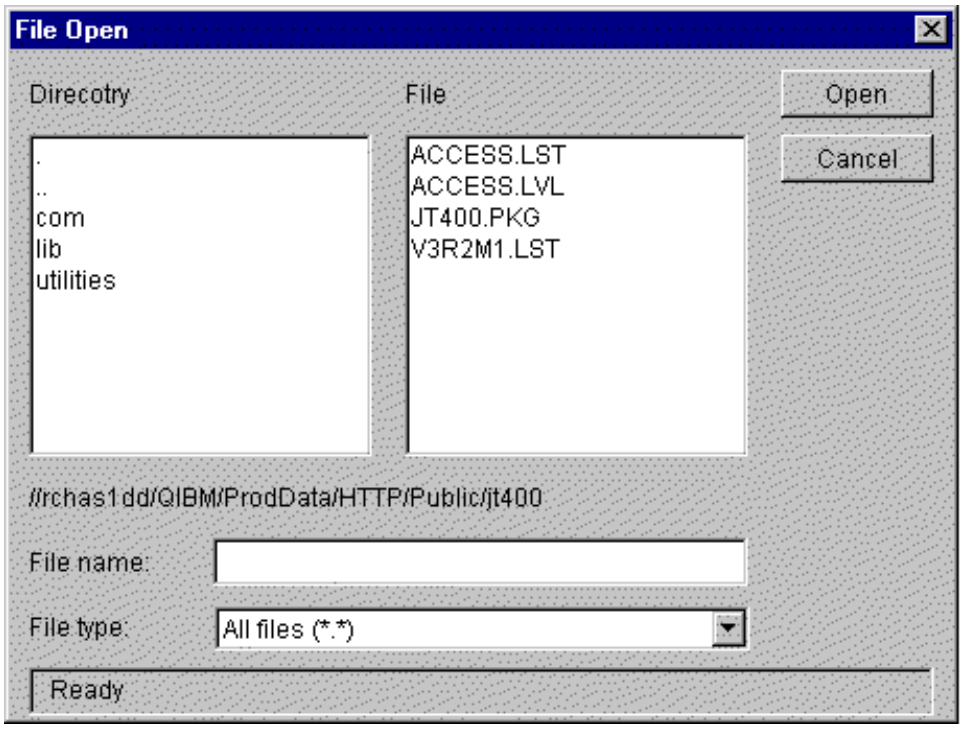
```
                                      // Creare un oggetto IFSFileDialog
                                      // impostando il testo della barra del titolo.
                    // Supponiamo che "system" sia un oggetto AS400
                // e la "frame" è una JFrame
                                       // creata ed inizializzata altrove.
IFSFileDialog dialog = new IFSFileDialog (frame, "Select a file", system);

                                      // Impostare un elenco di filtri per la finestra di dialogo.
                                      // Il primo filtro verrà utilizzato quando viene
                                      // visualizzata la prima finestra di dialogo.
    FileFilter[] filterList = {new FileFilter("All files (*.*)", "*.*"),
                     new FileFilter("HTML files (*.HTML", "*.HTM")};
                      // Poi, impostare i filtri nella finestra di dialogo.
    dialog.setFileFilter (filterList, 0);

                                      // Impostare il testo sui pulsanti.
    dialog.setOkButtonText("Open");
    dialog.setCancelButtonText("Cancel");

                // Visualizzare la finestra di dialogo. Se l'utente ha
                                      // selezionato un file premendo il pulsante
                // "Apri", quindi stampare il nome percorso del
                                      // file selezionato.
    if (dialog.showDialog () == IFSFileDialog.OK)
            System.out.println(dialog.getAbsolutePath());
```

Esempio

Presentare un IFSFileDialog e stampare la selezione, se presente.

La Figura 1 visualizza il componente della GUI (Graphical User Interface) IFSFileDialog:

Figura 1: componente GUI IFSFileDialog



Indirizzari in AS400Panes:

AS400Panes sono componenti GUI che presentano e consentono la gestione di una o più risorse del server.Un oggetto VIFSDirectory è una risorsa che rappresenta un indirizzario nell'IFS (Integrated File System) da utilizzare in AS400Panes.Gli oggetti AS400Pane e VIFSDirectory possono essere utilizzati insieme per presentare diverse viste dell'IFS (Integrated File System) e per consentire all'utente di scorrere, gestire e selezionare gli indirizzari e i file.

Per utilizzare un VIFSDirectory, impostare le proprietà sistema e percorso. E' possibile impostare tali proprietà utilizzando un constructor o tramite i metodi setSystem() e setPath(). Si include quindi l'oggetto VIFSDirectory in AS400Pane come root, utilizzando il constructor o il metodo setRoot() dell'AS400Pane.

VIFSDirectory presenta altre utili proprietà per la definizione di indirizzari e di file presentati in AS400Panes. Utilizzare setInclude() per specificare se vengono visualizzati gli indirizzari, i file o entrambi. Utilizzare setPattern() per impostare un filtro sulle voci visualizzate selezionando un modello a cui deve corrispondere il nome del file. E' possibile utilizzare caratteri jolly, come "*" e "?", nei modelli. Allo stesso modo, utilizzare setFilter() per impostare un filtro con un oggetto IFSFileFilter.

Quando vengono creati gli oggetti AS400Pane e VIFSDirectory, vengono inizializzati in uno stato predefinito. Gli indirizzari secondari e i file che costituiscono il contenuto dell'indirizzario root non sono stati caricati. Per caricare il contenuto, il programma di chiamata deve esplicitamente richiamare il metodo load() per entrambi gli oggetti al fine di inizializzare la comunicazione con il server per raccogliere il contenuto dell'indirizzario.

Durante l'esecuzione, un utente può eseguire operazioni su tutti gli indirizzari o file facendo clic con il tastino destro del mouse su di esso per visualizzare il menu contesto. Il menu contesto dell'indirizzario può includere le voci seguenti:

- **Crea file** - crea un file nell'indirizzario.Ciò fornirà al file un nome predefinito
- **Crea indirizzario** - crea un indirizzario secondario con un nome predefinito
- **Rinomina** - rinomina un indirizzario
- **Cancella** - cancella un indirizzario

- **Proprietà** - visualizza le proprietà come l'ubicazione, il numero di file e gli indirizzari secondari e la data di modifica

Il menu contesto del file può includere le voci seguenti:
- **Modifica** - modifica un file di testo in una finestra diversa
- **Visualizza** - visualizza un file di testo in una finestra diversa
- **Rinomina** - rinomina un file
- **Cancella** - cancella un file
- **Proprietà** - visualizza le proprietà come l'ubicazione, la dimensione, la data di modifica e gli attributi

Gli utenti possono solamente leggere o scrivere in indirizzari o file per cui dispongono di autorizzazione. Inoltre, è possibile che il programma di chiamata impedisca all'utente di eseguire le operazioni utilizzando il metodo setAllowActions() sul pannello.

L'esempio seguente crea un VIFSDirectory e lo visualizza in un AS400ExplorerPane:

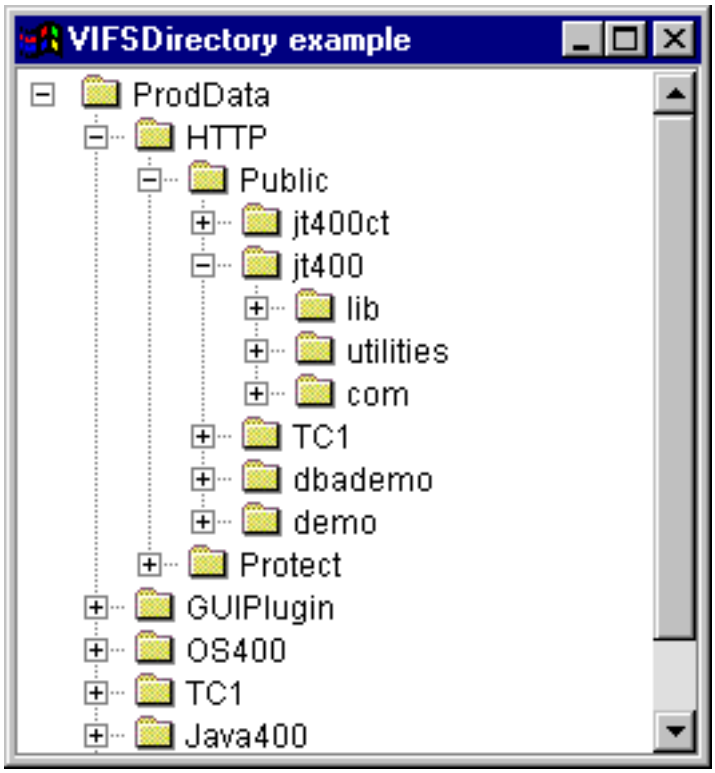
```
                                        // Creare l'oggetto VIFSDirectory.
                    // Supponiamo che "system" sia un oggetto AS400
                        // creato ed inizializzato
                        // da qualche altra parte.
VIFSDirectory root = new VIFSDirectory (system, "/DirectoryA/DirectoryB");

                                        // Creare e caricare un oggetto AS400ExplorerPane.
     AS400ExplorerPane explorerPane = new AS400ExplorerPane (root);
            explorerPane.load ();

                                        // Aggiungere il pannello explorer ad una frame.
                                        // Supponiamo che "frame" sia un JFrame
                    // creato da qualche altra parte.
    frame.getContentPane().add(explorerPane);
```

Esempio

Visualizzare una gerarchia indirizzario IFS (Integrated File System) utilizzando un AS400TreePane con un oggetto VIFSDirectory.

La Figura 1 visualizza il componente GUI (Graphical User Interface) VIFSDirectory:

Figura 1: Componente GUI VIFSDirectory



IFSTextFileDocument:

I documenti file di testo consentono a un programma Java di utilizzare tutti i componenti testo grafico JFC (Java Foundation Classes) per la modifica o la visualizzazione di file di testo nell'IFS (Integrated File System) su un server. (Un componente testo è un componente grafico utilizzato per visualizzare testo modificabile in via facoltativa dall'utente.)

La classe IFSTextFileDocument è un'implementazione dell'Interfaccia documento JFC.Può essere utilizzato direttamente con qualsiasi componente testo grafico JFC. Diversi componenti di testo, come i campi a riga singola (JTextField) e le aree testo a più righe (JTextArea), sono disponibili in JFC.

I documenti file di testo associano il contenuto di un componente testo a un file di testo. Il programma Java può effettuare il caricamento e il salvataggio tra il componente testo e il file di testo in qualsiasi momento.

Per utilizzare un IFSTextFileDocument, impostare le proprietà sistema e percorso. E' possibile impostare tali proprietà utilizzando un programma di creazione o tramite i metodi setSystem() e setPath(). L'oggetto IFSTextFileDocument viene quindi "collegato" al componente testo, in genere utilizzando il programma di creazione del componente testo o il metodo setDocument().

Inizialmente, il contenuto del componente testo è vuoto. Utilizzare load() per caricare il contenuto dal file di testo. Utilizzare save() per salvare il contenuto del componente testo nella coda dati.

L'esempio seguente crea e carica un IFSTextFileDocument:

```
                    // Creare e caricare l'oggetto
                    // Oggetto IFSTextFileDocument. Supponiamo
                    // che "system" sia un oggetto AS400 creato
                    // ed inizializzata altrove.
IFSTextFileDocument ifsDocument = new IFSTextFileDocument (system, "/DirectoryA/MyFile.txt");
     ifsDocument.load ();

                                        // Creare un'area di testo per presentare il
```

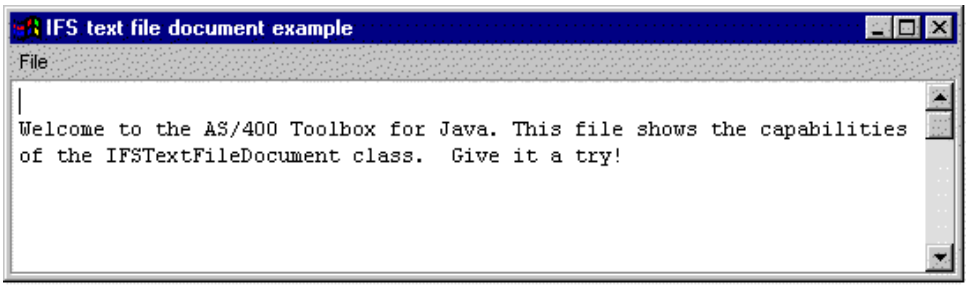
```
                                        // documento.
JTextArea textArea = new JTextArea (ifsDocument);

                                        // Aggiungere l'area di testo ad una frame.
                                        // Supponiamo che "frame" sia un JFrame
                                        // creato da qualche altra parte.
frame.getContentPane ().add (textArea);
```

Esempio

Visualizzare un IFSTextFileDocument in un JTextPane.

La Figura 1 visualizza il componente GUI (Graphical User Interface) IFSTextFileDocument:

Figura 1: esempio di IFSTextFileDocument



Classe VJavaApplicationCall

La classe VJavaApplicationCall consente di eseguire un'applicazione Java sul server da un client utilizzando una GUI (graphical user interface).

La GUI è un pannello con due sezioni. La sezione superiore è una finestra di emissione che visualizza l'emissione che il programma Java scrive nello standard output e nello standard error. La sezione inferiore è un campo di immissione in cui l'utente inserisce l'ambiente Java, il programma Java da eseguire con i parametri e l'immissione che il programma Java riceve attraverso lo standard input. Consultare Opzioni comando Java per ulteriori informazioni.

Ad esempio, questo codice potrebbe creare la seguente GUI per il programma Java.

VJavaApplicationCall è una classe che si richiama dal programma Java. Tuttavia, IBM Toolbox per Java fornisce anche un programma di utilità che è un'applicazione completa Java utilizzabile per richiamare il programma Java da una stazione di lavoro. Consultare la classe RunJavaApplication per ulteriori informazioni.

Classi JDBC

I componenti GUI (Graphical User Interface) JDBC consentono a un programma Java di visualizzare varie viste e controlli per l'accesso a un database utilizzando istruzioni e interrogazioni SQL (Structured Query Language).

I seguenti componenti sono disponibili:

- SQLStatementButton e SQLStatementMenuItem sono un pulsante o una voce di menu che invia un'istruzione SQL quando si seleziona o si fa clic su di essa.
- SQLStatementDocument è un documento utilizzabile con qualsiasi componente testo grafico JFC (Java Foundation Classes) per emettere un'istruzione SQL.
- SQLResultSetFormPane visualizza i risultati di un'interrogazione SQL in un modulo.

- SQLResultSetTablePane visualizza i risultati di un'interrogazione SQL in una tabella.
- SQLResultSetTableModel gestisce i risultati di un'interrogazione SQL in una tabella.
- SQLQueryBuilderPane mostra uno strumento interattivo per la creazione dinamica di interrogazioni SQL.

Tutti i componenti GUI (Graphical User Interface) JDBC comunicano con il database utilizzando una unità di controllo JDBC. L'unità di controllo JDBC deve essere registrata con il gestore unità di controllo JDBC per il corretto funzionamento di tutti i componenti. Il seguente esempio registra l'unità di controllo JDBC IBM Toolbox per Java:

```
                        // Registrare l'unità di controllo JDBC.
    DriverManager.registerDriver(new com.ibm.as400.access.AS400JDBCDriver());
```

Collegamenti SQL

Un oggettoSQLConnection rappresenta un collegamento a un database mediante JDBC.**L'oggetto SQLConnection viene utilizzato con tutti i componenti GUI (Graphical User Interface) JDBC.**

Per utilizzare un SQLConnection, impostare la proprietà URL utilizzando il programma di creazione o setURL().Ciò identifica il database verso cui viene effettuato il collegamento. E' possibile impostare altre proprietà facoltative:

- Utilizzare setProperties() per specificare una serie di collegamenti JDBC.
- Utilizzare setUserName() per specificare il nome utente del collegamento.
- Utilizzare setPassword() per specificare la parola d'ordine del collegamento.

Il collegamento effettivo al database non viene effettuato alla creazione dell'oggetto SQLConnection. Invece, viene effettuato quando viene richiamato getConnection().Questo metodo viene richiamato normalmente in maniera automatica dai componenti GUI (Graphical User Interface) JDBC, ma può essere richiamato in qualsiasi momento per controllare quando viene effettuato il collegamento.

L'esempio seguente crea e inizializza un oggetto SQLConnection:

```
                                        // Creare un oggetto SQLConnection.
SQLConnection connection = new SQLConnection ();

                                        // Impostare le proprietà URL e nome utente del collegamento.
connection.setURL ("jdbc:as400://MySystem");
connection.setUserName ("Lisa");
```

Un oggetto SQLConnection può essere utilizzato per più di un componente GUI (Graphical User Interface) JDBC. Tutti i componenti utilizzeranno lo stesso collegamento, il che può migliorare le prestazioni e l'utilizzo delle risorse. In alternativa, ogni componente GUI (Graphical User Interface) JDBC può utilizzare un diverso oggetto SQL. E' in qualche modo necessario utilizzare collegamenti separati, in modo tale che le istruzioni SQL vengano emesse in diverse transazioni.

Quando il collegamento non è più necessario, chiudere l'oggetto SQLConnection utilizzando close().Ciò libera risorse JDBC sul client e sul server.

Pulsanti e voci di menu:

Un oggetto SQLStatementButton rappresenta un pulsante che, quando viene premuto, emette un'istruzione SQL (Structured Query Language).La classe SQLStatementButton estende la classe JButton di JFC (Java Foundation Classes) in modo che tutti i pulsanti abbiano un aspetto e una funzionalità coerenti.

Allo stesso modo, un oggetto SQLStatementMenuItem rappresenta una voce di menu che, quando viene selezionata, emette un'istruzione SQL.La classe SQLStatementMenuItem estende la classe JFC JMenuItem in modo che tutte le voci di menu abbiano un aspetto e una funzionalità coerenti.

Per utilizzare una di queste classi, impostare il collegamento e le proprietà SQLStatement. Queste proprietà possono essere impostate utilizzando un programma di creazione o i metodi setConnection() e setSQLStatement().

Il seguente esempio crea un SQLStatementButton. Quando il pulsante viene premuto al momento dell'esecuzione, esso cancella tutti i record nella tabella:

```
                    // Creare un oggetto SQLStatementButton.
                                       // Il testo del pulsante cita "Cancella tutto",
                                       // e non vi è icona.
        SQLStatementButton button = new SQLStatementButton ("Delete All");

                                      // Impostare le proprietà di connection e
                                      // SQLStatement. Supponiamo che "connection"
                                      // sia un oggetto SQLConnection creato
                                      // ed inizializzata altrove.
        button.setConnection (connection);
        button.setSQLStatement ("DELETE FROM MYTABLE");

                    // Aggiungere il pulsante ad una frame. Supponiamo
                    // che "frame" sia un JFrame creato da qualche altra
                    // da qualche altra parte.
        frame.getContentPane ().add (button);
```

Dopo l'emissione di un'istruzione SQL, utilizzare getResultSet(), getMoreResults(), getUpdateCount() o getWarnings() per richiamare i risultati.

Classe SQLStatementDocument:

La classe SQLStatementDocument è un'implementazione dell'interfaccia del documento JFC (Java Foundation Classes).Può essere utilizzato direttamente con qualsiasi componente testo grafico JFC. Alcuni componenti di testo, come campi a riga singola (JTextField) e aree di testo a più righe (JTextArea), sono disponibili in JFC. Gli oggetti SQLStatementDocument associano il contenuto dei componenti di testo agli oggetti SQLConnection. Il programma Java può eseguire l'istruzione SQL presente nel documento in ogni momento e poi elaborare i risultati, nel caso in cui siano presenti.

Per utilizzare un SQLStatementDocument, impostare la proprietà del collegamento. Impostare questa proprietà utilizzando il programma di creazione o il metodo setConnection(). L'oggetto SQLStatementDocument viene quindi ″collegato″ al componente testo, in genere utilizzando il programma di creazione del componente testo o il metodo setDocument(). Utilizzare execute() in qualsiasi momento per eseguire un'istruzione SQL contenuta nel documento.

Il seguente esempio crea un SQLStatementDocument in un JTextField:

```
                                      // Creare un oggetto SQLStatementDocument.
                                      // AS400TreePane.
Supponiamo che "connection"
                                      // sia un oggetto SQLConnection
                                      // creato ed inizializzato altrove.
                                      // Il testo del documento è
                                      // inizializzato in una interrogazione generica.
        SQLStatementDocument document = new SQLStatementDocument (connection, "SELECT * FROM QIWS.QCUSTCDT");

                                      // Creare un campo testo per presentare il
                                      // documento.
        JTextField textField = new JTextField ();
        textField.setDocument (document);

                                      // Aggiungere il campo testo ad una frame.
```

```
                                        // Supponiamo che "frame" sia un JFrame
                        // creato da qualche altra parte.
        frame.getContentPane ().add (textField);

                                        // Eseguire l'istruzione SQL contenuta
                                        // nel campo testo.
                                document.execute ();
```

Dopo l'emissione dell'istruzione SQL, utilizzare getResultSet(), getMoreResults(), getUpdateCount() o getWarnings() per richiamare i risultati.

Classe SQLResultSetFormPane:

Un SQLResultSetFormPane presenta i risultati dell'interrogazione SQL (Structured Query Language) in un modulo.Il modulo visualizza un record alla volta e fornisce i pulsanti che consentono all'utente di scorrere avanti, indietro, al primo o all'ultimo record oppure di aggiornare il pannello dei risultati.

Per utilizzare un SQLResultSetFormPane, impostare le proprietà del collegamento e dell'interrogazione. Impostare queste proprietà utilizzando il programma di creazione o i metodi setConnection() e setQuery().Utilizzare load() per eseguire l'interrogazione e presentare il primo record nell'impostazione risultato.Quando i risultati non sono più necessari, richiamare close() per assicurarsi che la serie di risultati sia chiusa.

Il seguente esempio crea un oggetto SQLResultSetFormPane e lo aggiunge ad un segmento:

```
                                        // Creare un oggetto SQLResultSetFormPane.
                                        // AS400TreePane.
Supponiamo che "connection"
                                        // sia un oggetto SQLConnection
                                        // creato ed inizializzato altrove.
     SQLResultSetFormPane formPane = new SQLResultSetFormPane (connection, "
                                     SELECT * FROM QIWS.QCUSTCDT");
                                        // Caricare i risultati.
            formPane.load ();

                                        // Aggiungere il pannello modulo ad una frame.
                                        // Supponiamo che "frame" sia un JFrame
                        // creato da qualche altra parte.
        frame.getContentPane ().add (formPane);
```

Classe SQLResultSetTablePane:

Un SQLResultSetTablePane presenta i risultati di un'interrogazione SQL (Structured Query Language) in una tabella.Ogni riga nella tabella visualizza un record dalla serie di risultati e ogni colonna visualizza un campo.

Per utilizzare un SQLResultSetTablePane, impostare le proprietà del collegamento e dell'interrogazione. Impostare le proprietà utilizzando il programma di creazione o i metodi setConnection() e setQuery().Utilizzare load() per eseguire l'interrogazione e presentare i risultati nella tabella.Quando i risultati non sono più necessari, richiamare close() per assicurarsi che la serie di risultati sia chiusa.

Il seguente esempio crea un oggetto SQLResultSetTablePane e lo aggiunge ad un segmento:

```
                                        // Creare un oggetto SQLResultSetTablePane.
                                        // AS400TreePane.
Supponiamo che "connection"
                                        // sia un oggetto SQLConnection
                                        // creato ed inizializzato altrove.
     SQLResultSetTablePane tablePane = new SQLResultSetTablePane (connection,
                                      "SELECT * FROM QIWS.QCUSTCDT");
                                        // Caricare i risultati.
                        tablePane.load ();
```

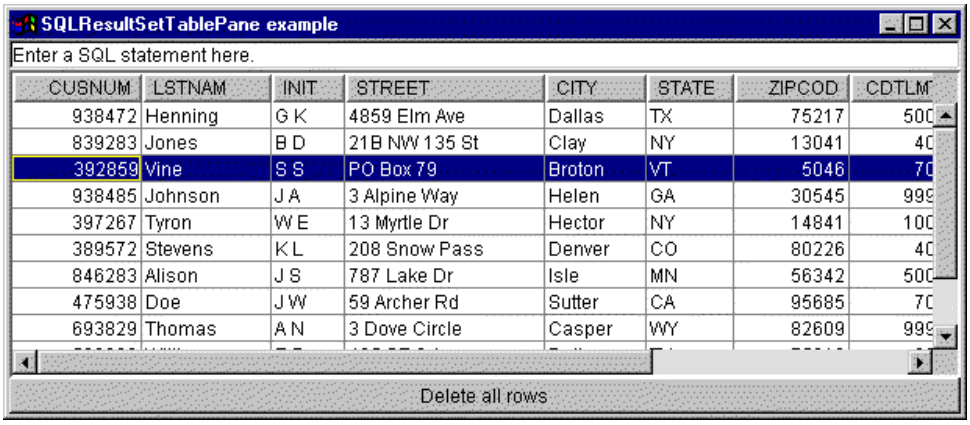
```
                                        // Aggiungere il pannello tabella ad una frame.
                                        // Supponiamo che "frame" sia un JFrame
                    // creato da qualche altra parte.
frame.getContentPane ().add (tablePane);
```

Esempio

Presentare un SQLResultSetTablePane che visualizza il contenuto di una tabella.Questo esempio utilizza un SQLStatementDocument (indicato nella seguente immagine dal testo: ″Enter a SQL statement here″) che consente all'utente di immettere ogni istruzione SQL e un SQLStatementButton (indicato dal testo: ″Delete all rows″) che consente all'utente di cancellare tutte le righe dalla tabella.

La figura 1 visualizza il componente GUI SQLResultSetTablePane:

Figura 1: componente GUI SQLResultSetTablePane



Classe SQLResultSetTableModel:

SQLResultSetTablePane viene implementato utilizzando il paradigma programma di controllo vista modello, nel quale i dati e l'interfaccia del cliente sono separati in classi diverse. L'implementazione integra SQLResultSetTableModel con JTable di JFC (Java Foundation Classes').La classe SQLResultSetTableModel gestisce i risultati dell'interrogazione e JTable visualizza graficamente i risultati e gestisce l'interazione utente.

SQLResultSetTablePane fornisce sufficiente funzionalità per la maggior parte dei requisiti. Tuttavia, se il programma di chiamata richiede un ulteriore controllo del componente JFC, potrà utilizzare direttamente SQLResultSetTableModel e fornire un'integrazione personalizzata con un componente GUI (graphical user interface) diverso.

Per utilizzare un SQLResultSetTableModel, impostare il collegamento e le proprietà dell'interrogazione.Impostare queste proprietà utilizzando il programma di creazione o i metodi setConnection() e setQuery().Utilizzare load() per eseguire l'interrogazione e caricare i risultati.Quando i risultati non sono più necessari, richiamare close() per assicurarsi che la serie di risultati sia chiusa.

Il seguente esempio crea un oggetto SQLResultSetTableModel e lo presenta con JTable:

```
                    // Creare un oggetto SQLResultSetTableModel.
                                        // AS400TreePane.
Supponiamo che "connection"
                                        // sia un oggetto SQLConnection
                                        // creato ed inizializzato altrove.
    SQLResultSetTableModel tableModel = new SQLResultSetTableModel (connection,
                                       "SELECT * FROM QIWS.QCUSTCDT");
                                        // Caricare i risultati.
```

```
tableModel.load ();

                                    // Creare una JTable per il modello.
JTable table = new JTable (tableModel);

             // Aggiungere la tabella ad una frame. Supponiamo
             // che "frame" sia un JFrame creato da qualche altra
             // da qualche altra parte.
frame.getContentPane ().add (table);
```

Programmi di creazione di interrogazione SQL:

Un SQLQueryBuilderPane presenta uno strumento interattivo per la creazione dinamica delle interrogazioni SQL.

Per utilizzare un SQLQueryPane, impostare la proprietà di collegamento. E' possibile impostare questa proprietà utilizzando il programma di creazione o il metodo setConnection().Utilizzare load() per caricare i dati necessari per la GUI del programma di creazione dell'interrogazione.Utilizzare getQuery() per richiamare l'interrogazione SQL creata dall'utente.

Il seguente esempio crea un oggetto SQLQueryBuilderPane e lo aggiunge ad una frame:

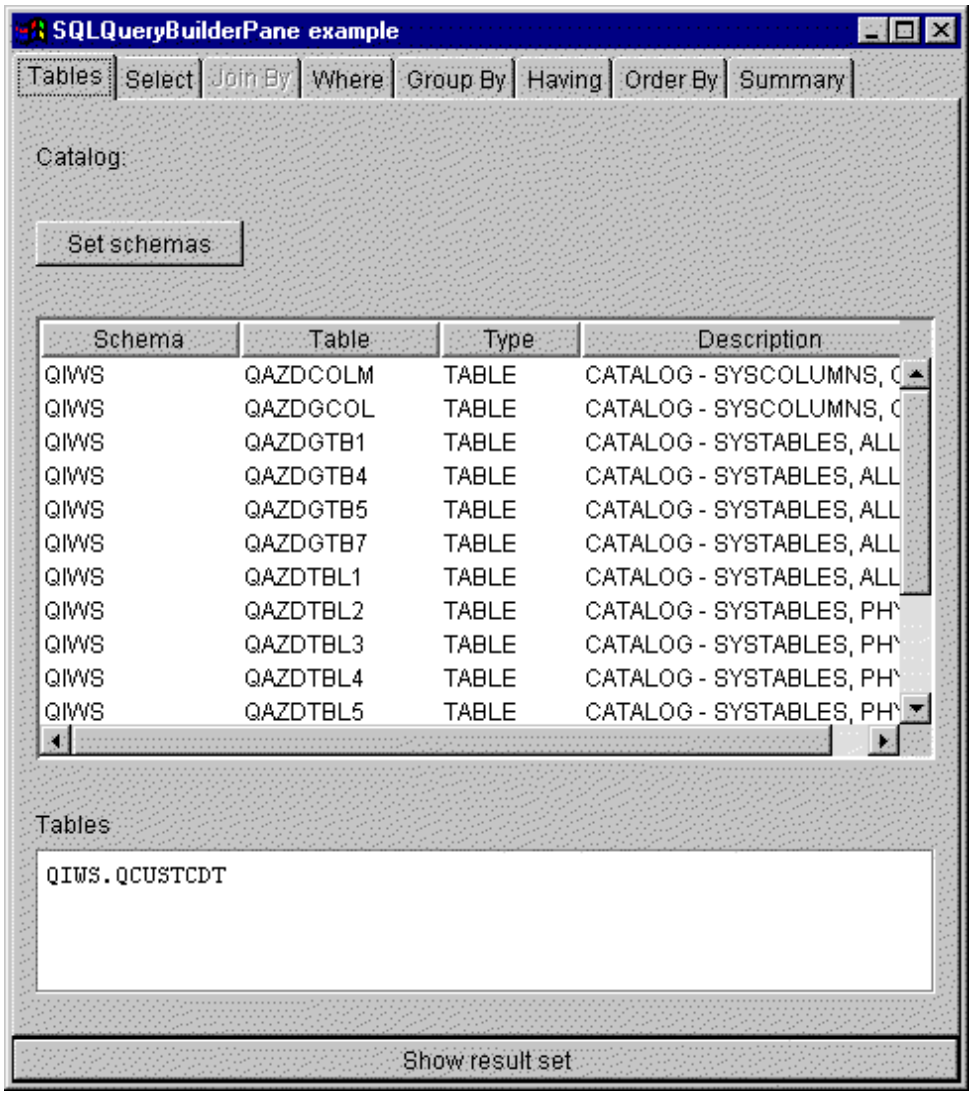
```
                                    // Creare un oggetto SQLQueryBuilderPane.
                                    // AS400TreePane.
Supponiamo che "connection"
                                    // sia un oggetto SQLConnection
                                    // creato ed inizializzato altrove.
     SQLQueryBuilderPane queryBuilder = new SQLQueryBuilderPane (connection);

                                    // Caricare i dati necessari per il programma di creazione
                                    // dell'interrogazione.
queryBuilder.load ();

             // Aggiungere il pannello programma di creazione della query ad una
             // frame. Supponiamo che "frame" sia un oggetto
                                    // JFrame creato altrove.
frame.getContentPane ().add (queryBuilder);
```

Esempio

Presentare un SQLQueryBuilderPane e un pulsante.Quando si fa clic sul pulsante, esso presenta i risultati dell'interrogazione in un SQLResultSetFormPane in un'altra frame.

La figura 1 illustra il componente GUI SQLQueryBuilderPane:

Figura 1: componente GUI SQLQueryBuilderPane



Lavori

I componenti vaccess (GUI) dei lavori consentono ad un programma Java di presentare gli elenchi dei lavori del server e dei messaggi di registrazione lavori in una GUI.

I seguenti componenti sono disponibili:

- Un oggetto VJobList è una risorsa che rappresenta un elenco di lavori del server da utilizzare in AS400Panes.
- Un oggetto VJob è una risorsa che rappresenta l'elenco dei messaggi nella registrazione lavori da utilizzare in AS400Panes.

E' possibile utilizzare insieme gli oggetti AS400Panes, VJobList e VJob per presentare molte viste di un elenco di lavori o di una registrazione lavori.

Per utilizzare un VJobList, impostare le proprietà sistema, nome, numero e utente. Impostare queste proprietà utilizzando un programma di creazione oppure attraverso le proprietà setSystem(), setName(), setNumber() e setUser().

Per utilizzare un VJob, impostare la proprietà di sistema. Impostare questa proprietà utilizzando un programma di creazione o il metodo setSystem().

L'oggetto VJobList o VJob viene poi collegato a AS400Pane come root, utilizzando il programma di creazione del pannello o il metodo setRoot().

VJobList ha altre proprietà utili per definire la serie dei lavori presentati in AS400Panes. Utilizzare setName() per specificare che si visualizzano solo lavori con un a certo nome. Utilizzare setNumber() per specificare che si visualizzano solo lavori con un certo numero. In modo analogo, utilizzare setUser() per specificare che si visualizzano solo lavori relativi ad un certo utente.

Quando vengono creati, gli oggetti AS400Pane, VJobList e VJob, sono inizializzati su uno stato predefinito. L'elenco di lavori o dei messaggi di registrazione lavori non vengono caricati al momento della creazione. Per caricare il contenuto, è necessario che il programma di chiamata richiami esplicitamente il metodo load() su entrambi gli oggetti. Ciò avvierà la comunicazione al server per raccogliere il contenuto dell'elenco.

Al momento dell'esecuzione, fare clic con il tastino destro del mouse su un lavoro, un elenco lavori o un messaggio di registrazione lavori per visualizzare il menu di scelta rapida. Selezionare **Proprietà** dal menu di scelta rapida per eseguire le azioni sull'oggetto selezionato:

- Lavoro - Gestire la proprietà, come il tipo e lo stato. E' inoltre possibile modificare il valore di alcune delle proprietà.
- Elenco lavori - Gestire la proprietà, come il nome, numero e proprietà utente. E' inoltre possibile modificare il contenuto dell'elenco.
- Messaggio registrazione lavori - Visualizzare le proprietà, come il testo completo, severità e tempo di trasmissione.

Gli utenti possono accedere solo ai lavori a cui sono autorizzati. Inoltre, è possibile che il programma Java impedisca all'utente di eseguire le operazioni utilizzando il metodo setAllowActions() sul pannello.

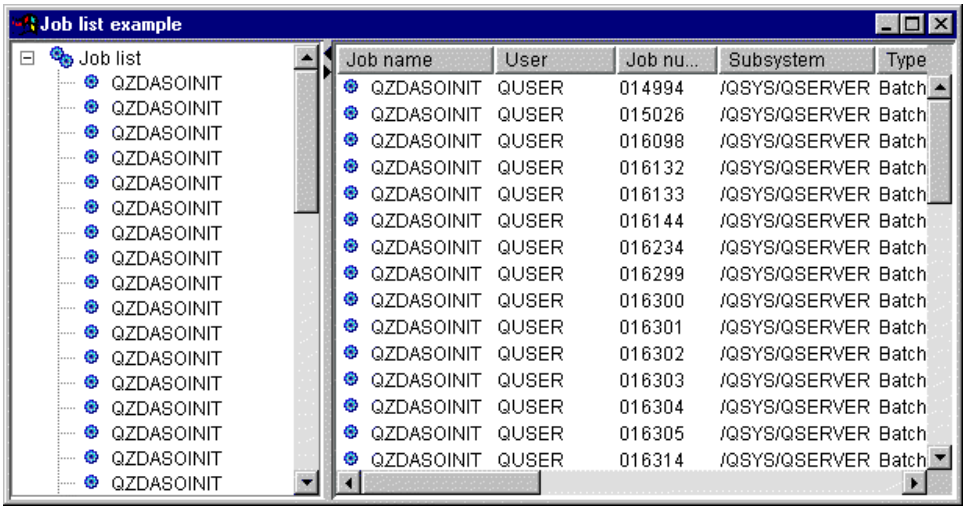
Il seguente esempio crea VJobList e lo presenta in AS400ExplorerPane:

```
                    // Creare l'oggetto VJobList. Supponiamo
                                        // che "system" sia un oggetto AS400
                                        // creato ed inizializzato altrove.
          VJobList root = new VJobList (system);

                        // Creare e caricare un
                                        // oggetto AS400ExplorerPane.
          AS400ExplorerPane explorerPane = new AS400ExplorerPane (root);
                 explorerPane.load ();

                                        // Aggiungere il pannello explorer ad una frame.
                                        // Supponiamo che "frame" sia un JFrame
                        // creato da qualche altra parte.
        frame.getContentPane().add(explorerPane);
```

Esempi

Questo esempio di VJobList presenta un AS400ExplorerPane contenente un elenco di lavori. L'elenco visualizza i lavori sul sistema che hanno lo stesso nome lavoro.

La seguente immagine visualizza il componente GUI VJobList:



Classi di messaggio Vaccess

I componenti GUI dei messaggi consentono a un programma Java di presentare gli elenchi dei messaggi del server nella GUI.

I seguenti componenti sono disponibili:

- Un oggetto Elenco messaggi è una risorsa che rappresenta un elenco di messaggi da utilizzare in AS400Panes. Ciò viene creato per gli elenchi dei messaggi da un comando o dalle chiamate al programma.
- Un oggetto Coda messaggi è una risorsa che rappresenta i messaggi in una coda messaggi del server da utilizzare in AS400Ppanes.

AS400Panes sono componenti della GUI che presentano e consentono una gestione di una o più risorse del server. Gli oggetti VMessageList e VMessageQueue sono risorse che presentano gli elenchi dei messaggi del server in AS400Panes.

E' possibile utilizzare gli oggetti AS400Pane, VMessageList e VMessageQueue insieme per presentare molte viste di un elenco dei messaggi e per consentire agli utenti di selezionare ed eseguire le operazioni sui messaggi.

Classe VMessageList:

Un oggetto VMessageList è una risorsa che rappresenta un elenco di messaggi da utilizzare in AS400Panes. Ciò viene creato per gli elenchi dei messaggi da un comando o dalle chiamate al programma. Il seguente metodo restituisce gli elenchi di messaggi:

- CommandCall.getMessageList()
- CommandCallButton.getMessageList()
- CommandCallMenuItem.getMessageList()
- ProgramCall.getMessageList()
- ProgramCallButton.getMessageList()
- ProgramCallMenuItem.getMessageList()

Per utilizzare un VMessageList, impostare la proprietà messageList. Impostare questa proprietà utilizzando un programma di creazione o attraverso il metodo setMessageList().L'oggetto VMessageList viene quindi collegato a AS400Pane come root, utilizzando il programma di creazione o il metodo setRoot() di AS400Pane.

Quando vengono creati gli oggetti AS400Pane e VMessageList, vengono inizializzati su uno stato predefinito. L'elenco dei messaggi non viene caricato al momento della creazione. Per caricare il contenuto, è necessario che il programma di chiamata richiami esplicitamente il metodo load() su entrambi gli oggetti.

Al momento dell'esecuzione, un utente può eseguire le azioni sul messaggio facendovi clic col tastino destro del mouse per visualizzare il menu di contesto. Il menu di contesto del messaggio può contenere una voce denominata **Proprietà** che visualizza le proprietà come la severità, il tipo e la data.

E' possibile che il programma di chiamata impedisca all'utente di eseguire le azioni utilizzando il metodo setAllowActions() sul pannello.

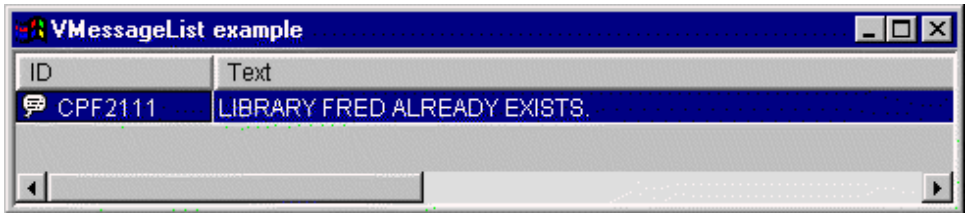
Il seguente esempio crea un VMessageList per i messaggi creati da un comando di chiamata e lo presenta in un AS400DetailsPane:

```
                                    // Creare l'oggetto VMessageList.
                // Supponiamo che "command" sia
                                    // un oggetto CommandCall creato ed eseguito
                    // da qualche altra parte.
     VMessageList root = new VMessageList (command.getMessageList ());

                                    // Creare e caricare un oggetto
                                    // AS400TreePane.
    AS400DetailsPane detailsPane = new AS400DetailsPane (root);
            detailsPane.load ();

                    // Aggiungere il pannello dettagli ad una frame.
                                    // Supponiamo che "frame" sia un JFrame
                    // creato da qualche altra parte.
frame.getContentPane().add(detailsPane);
```

Esempio

Presentare l'elenco dei messaggi creati da un comando di chiamata utilizzando un AS400DetailsPane con un oggetto VMessageList. La figura 1 visualizza il componente GUI VMessageList:

Figura 1: componente GUI VMessageList



Classe VMessageQueue:

Un oggetto VMessageQueue è una risorsa che rappresenta i messaggi in una coda messaggi del server da utilizzare in AS400Panes.

Per utilizzare una VMessageQueue, impostare il sistema e le proprietà percorso. Queste proprietà possono essere impostate utilizzando un programma di creazione o attraverso i metodi setSystem() e setPath().L'oggetto VMessageQueue viene quindi collegato a AS400Pane come root, utilizzando il programma di creazione o il metodo setRoot() di AS400Pane.

VMessageQueue ha altre proprietà utili per definire la serie dei messaggi che sono presenti in AS400Panes. Utilizzare setSeverity() per specificare la severità dei messaggi visualizzati. Utilizzare setSelection() per specificare il tipo di messaggio visuaòlizzato.

Quando vengono creati gli oggetti AS400Pane e VMessageQueue, sono inizializzati su uno stato predefinito. L'elenco dei messaggi non viene caricato al momento della creazione. Per caricare il contenuto, è necessario che il programma di chiamata richiami esplicitamente il metodo load() su entrambi gli oggetti. Ciò avvierà la comunicazione al server per raccogliere il contenuto dell'elenco.

Al momento dell'esecuzione, un utente può eseguire le azioni sul messaggio o sulla coda messaggi facendovi clic con il tastino destro del mouse per visualizzare il menu di contesto. Il menu di contesto per le code messaggi può contenere le seguenti voci:

- **Ripulisci** - elimina il contenuto della coda messaggi
- **Proprietà** - consente all'utente di impostare la severità e la selezione proprietà. Questo può essere utilizzato per modificare il contenuto dell'elenco

La seguente azione è disponibile per i messaggi sulla coda messaggi:

- **Elimina** - elimina il messaggio dalla coda messaggi
- **Rispondi** - risponde al messaggio di interrogazione
- **Proprietà** - visualizza le proprietà come la severità, il tipo e la data

Naturalmente, gli utenti possono accedere solo alle code messaggi a cui sono autorizzati. Inoltre, è possibile che il programma di chiamata impedisca all'utente di eseguire le operazioni utilizzando il metodo setAllowActions() sul pannello.

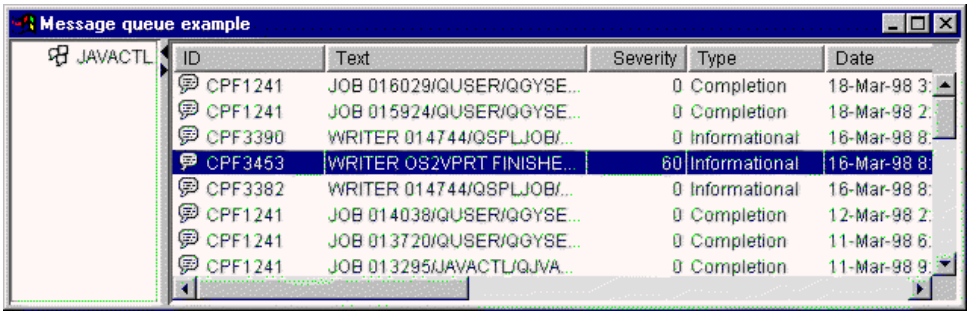
Il seguente esempio crea una VMessageQueue e la presenta in un AS400ExplorerPane:

```
                                        // Creare l'oggetto VMessageQueue.
                        // Supponiamo che "system" sia un oggetto AS400
                        // creato ed inizializzato
                        // da qualche altra parte.
    VMessageQueue root = new VMessageQueue (system, "/QSYS.LIB/MYLIB.LIB/MYMSGQ.MSGQ");

                        // Creare e caricare un
                                        // oggetto AS400ExplorerPane.
        AS400ExplorerPane explorerPane = new AS400ExplorerPane (root);
               explorerPane.load ();

                                        // Aggiungere il pannello explorer ad una frame.
                                        // Supponiamo che "frame" sia un JFrame
                        // creato da qualche altra parte.
        frame.getContentPane().add(explorerPane);
```

Esempio

Presentare l'elenco dei messaggi in una coda messaggi utilizzando un AS400ExplorerPane con un oggetto VMessageQueue. La figura 1 illustra il componente GUI VMessageQueue:

Figura 1: componente GUI VMessageQueue



Classi Permission

Le informazioni delle classi Permission possono essere utilizzate nella GUI attraverso le classi VIFSFile e VIFSDirectory.L'autorizzazione è stata aggiunta come un'azione in ognuna di questi classi.

Il seguente esempio visualizza come utilizzare l'autorizzazione con la classe VIFSDirectory:

```
// Creare l'oggetto AS400
AS400 as400 = new AS400();

// Creare un IFSDirectory utilizzando il nome di sistema
// ed il percorso completo di un oggetto QSYS
VIFSDirectory directory = new VIFSDirectory(as400,
                                                    "/QSYS.LID/testlib1.lib");

// Creare un Pannello explorer
AS400ExplorerPane pane = new AS400ExplorerPane(VNode)directory);

// Caricare le informazioni
pane.load();
```

Classi vaccess print

I seguenti componenti nel pacchetto vaccess consentono ad un programma Java di presentare elenchi di risorse di stampa del server nella GUI:

- Un oggetto VPrinters è una risorsa che rappresenta un elenco di stampanti da utilizzare in AS400Panes.
- Un oggetto VPrinter è una risorsa che rappresenta una stampante e i rispettivi file di spool da utilizzare in AS400Panes.
- Un oggetto VPrinterOutput è una risorsa che rappresenta un elenco di file di spool da utilizzare in AS400Panes.
- Un oggetto SpooledFileViewer è una risorsa che rappresenta visibilmente i file di spool.

AS400Panes sono componenti GUI che presentano e consentono la gestione di una o più risorse del server. Gli oggetti VPrinters, VPrinter e VPrinterOutput sono risorse che rappresentano gli elenchi delle risorse di stampa del server in AS400Panes.

E' possibile utilizzare insieme gli oggetti AS400Pane, VPrinters, VPrinter e VPrinterOutput per presentare diverse viste delle risorse di stampa e per consentire all'utente di selezionare e eseguire le operazioni su di esse.

Classe VPrinters:

Un oggetto VPrinters è una risorsa che rappresenta un elenco di stampanti da utilizzare in AS400Panes.

Per utilizzare un oggetto VPrinters, impostare la proprietà di sistema. Impostare questa proprietà utilizzando un programma di creazione o il metodo setSystem().L'oggetto VPrinters viene quindi collegato all'AS400Pane come root, utilizzando il programma di creazione del pannello o il metodo setRoot().

Un oggetto VPrinters ha un'altra proprietà utile per definire la serie di stampanti presentata in AS400Panes. Utilizzare setPrinterFilter() per specificare un filtro che definisce quali stampanti dovrebbero essere visualizzate.

Quando vengono creati gli oggetti AS400Pane e VPrinters, sono inizializzati su uno stato predefinito. L'elenco delle stampanti non è stato caricato. Per caricare il contenuto, è necessario che il programma di chiamata richiami esplicitamente il metodo load() su entrambi gli oggetti.

Al momento dell'esecuzione, un utente può eseguire le operazioni su ogni stampante o file di spool facendovi clic con il tastino destro del mouse per visualizzare il menu di contesto. Il menu di contesto dell'elenco stampanti può comprendere una voce denominata **Proprietà** che consente all'utente di impostare la proprietà di filtro della stampante che, a sua volta, può modificare il contenuto dell'elenco.

Il menu di contesto della stampante può comprendere le seguenti voci:

- **Congela** - congela la stampante
- **Rilascia** - rilascia la stampante
- **Avvia** - avvia la stampante
- **Arresta** - arresta la stampante
- **Rendi disponibile** - rende disponibile la stampante
- **Rendi non disponibile** - rende la stampante non disponibile
- **Proprietà** - visualizza le proprietà della stampante e consente all'utente di impostare i filtri

Gli utenti possono solo accedere alle stampanti a cui sono autorizzati. Inoltre, è possibile che il programma di chiamata impedisca all'utente di eseguire le operazioni utilizzando il metodo setAllowActions() sul pannello.

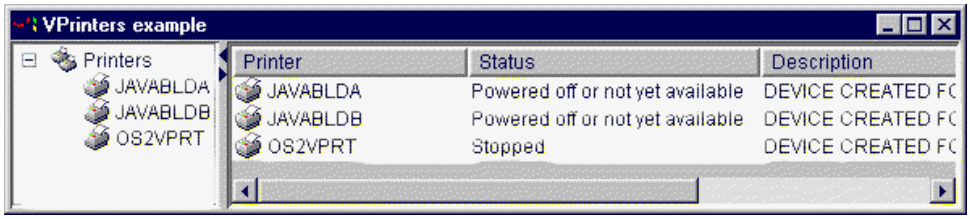
Il seguente esempio crea un oggetto VPrinters e lo presenta in un AS400TreePane

```
                                    // Creare l'oggetto VPrinters.
                  // Supponiamo che "system" sia un oggetto AS400
                  // creato ed inizializzato
                  // da qualche altra parte.
VPrinters root = new VPrinters (system);

                                    // Creare e caricare un oggetto
                                    // AS400TreePane.
AS400TreePane treePane = new AS400TreePane (root);
       treePane.load ();

                                    // Aggiungere un pannello albero ad una frame.
                                    // Supponiamo che "frame" sia un JFrame
                  // creato da qualche altra parte.
frame.getContentPane ().add (treePane);
```

Esempio

Presentare le risorse di stampa utilizzando unAS400ExplorerPane con un oggetto VPrinters.La figure 1 illustra il componente GUI VPrinters:

Figura 1: componente GUI VPrinters



Classe VPrinter:

Un oggetto VPrinter è una risorsa che rappresenta una stampante del server e i rispettivi file di spool da utilizzare in AS400Panes.

Per utilizzare VPrinter, impostare la proprietà della stampante. Impostare questa proprietà utilizzando un programma di creazione o il metodo setPrinter().L'oggetto VPrinter viene quindi collegato all'AS400Pane come root, utilizzando il programma di creazione del pannello o il metodo setRoot().

Quando vengono creati gli oggetti AS400Pane e VPrinter, sono inizializzati su uno stato predefinito. Gli attributi della stampante e l'elenco dei file di spool non vengono caricati al momento della creazione.

Per caricare il contenuto, è necessario che il programma di chiamata richiami esplicitamente il metodo load() su entrambi gli oggetti. Ciò avvierà la comunicazione al server per raccogliere il contenuto dell'elenco.

Al momento dell'esecuzione, un utente può eseguire le operazioni su ogni stampante o file di spool facendovi clic con il tastino destro del mouse per visualizzare il menu di contesto. Il menu di contesto per le code messaggi può contenere le seguenti voci:

- **Congela** - congela la stampante
- **Rilascia** - rilascia la stampante
- **Avvia** - avvia la stampante
- **Arresta** - arresta la stampante
- **Rendi disponibile** - rende disponibile la stampante
- **Rendi non disponibile** - rende la stampante non disponibile
- **Proprietà** - visualizza le proprietà della stampante e consente all'utente di impostare i filtri

Il menu di contesto per i file di spool elencati per una stampante può contenere le seguenti voci:

- **Rispondi** - risponde al file di spool
- **Congela** - congela il file di spool
- **Rilascia** - rilascia il file di spool
- **Stampa successivo** - stampa il successivo file di spool
- **Invia** - invia il file di spool
- **Sposta** - sposta il file di spool
- **Cancella** - cancella il file di spool
- **Proprietà** - visualizza molte proprietà del file di spool e consente all'utente di modificarne alcune

Gli utenti possono accedere solo alle stampanti e ai file di spool ai quali sono autorizzati. Inoltre, è possibile che il programma di chiamata impedisca all'utente di eseguire le operazioni utilizzando il metodo setAllowActions() sul pannello.

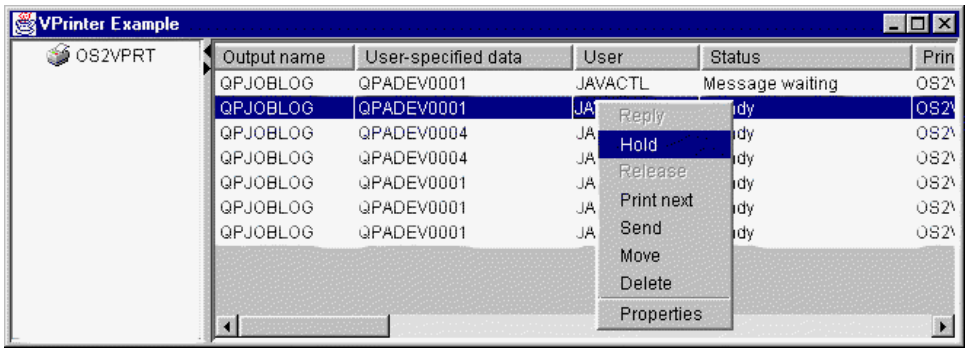
Il seguente esempio crea un VPrinter e lo presenta in un AS400ExplorerPane:

```
                                        // Creare l'oggetto VPrinter.
                        // Supponiamo che "system" sia un oggetto AS400
                        // creato ed inizializzato
                        // da qualche altra parte.
    VPrinter root = new VPrinter (new Printer (system, "MYPRINTER"));

                        // Creare e caricare un
                                        // oggetto AS400ExplorerPane.
         AS400ExplorerPane explorerPane = new AS400ExplorerPane (root);
                explorerPane.load ();

                                        // Aggiungere il pannello explorer ad una frame.
                                        // Supponiamo che "frame" sia un JFrame
                        // creato da qualche altra parte.
        frame.getContentPane().add(explorerPane);
```

Esempio

Presentare risorse di stampa utilizzando AS400ExplorerPane con un oggetto VPrinter.La figura 1 illustra il componente GUI VPrinter:

Figura 1: componente GUI VPrinter



Classe VPrinterOutput:

Un oggetto VPrinterOutput è una risorsa che rappresenta un elenco di file di spool sul server da utilizzare in AS400Panes.

Per utilizzare un oggetto VPrinterOutput, impostare la proprietà di sistema. E' possibile impostare questa proprietà utilizzando un programma di creazione o il metodo setSystem().L'oggetto VPrinterOutput viene quindi collegato all'AS400Pane come root, utilizzando il programma di creazione o il metodo setRoot() di AS400Pane.

Un oggetto VPrinterOutput ha altre proprietà utili per definire la serie dei file di spool presentata in AS400Panes. Utilizzare setFormTypeFilter() per specificare quali tipi di formati dovrebbero essere visualizzati.Utilizzare setUserDataFilter() per specificare quali dati utente dovrebbero essere visualizzati.Infine, utilizzare setUserFilter() per specificare quali file di spool degli utenti dovrebbero essere visualizzati.

Quando vengono creati gli oggetti AS400Pane e VPrinterOutput, sono inizializzati su uno stato predefinito. L'elenco dei file di spool non viene caricato al momento della creazione. Per caricare il contenuto, è necessario che il programma di chiamata richiami esplicitamente il metodo load() su entrambi gli oggetti. Ciò avvierà la comunicazione al server per raccogliere il contenuto dell'elenco.

Al momento dell'esecuzione, un utente può eseguire le operazioni su ogni file di spool o sull'elenco dei file di spool facendovi clic con il tastino destro del mouse per visualizzare il menu di contesto. E' possibile che il menu di contesto dell'elenco di file di spool comprenda una voce denominata **Proprietà** che consente all'utente di impostare le proprietà di filtro che possono modificare il contenuto dell'elenco.

Il menu di contesto del file di spool può comprendere le seguenti voci:
- **Rispondi** - risponde al file di spool
- **Congela** - congela il file di spool
- **Rilascia** - rilascia il file di spool
- **Stampa successivo** - stampa il successivo file di spool
- **Invia** - invia il file di spool
- **Sposta** - sposta il file di spool
- **Cancella** - cancella il file di spool
- **Proprietà** - visualizza molte proprietà del file di spool e consente all'utente di modificarne alcune

Naturalmente, gli utenti possono accedere solo ai file di spool a cui sono autorizzati. Inoltre, è possibile che il programma di chiamata impedisca all'utente di eseguire le operazioni utilizzando il metodo setAllowActions() sul pannello.

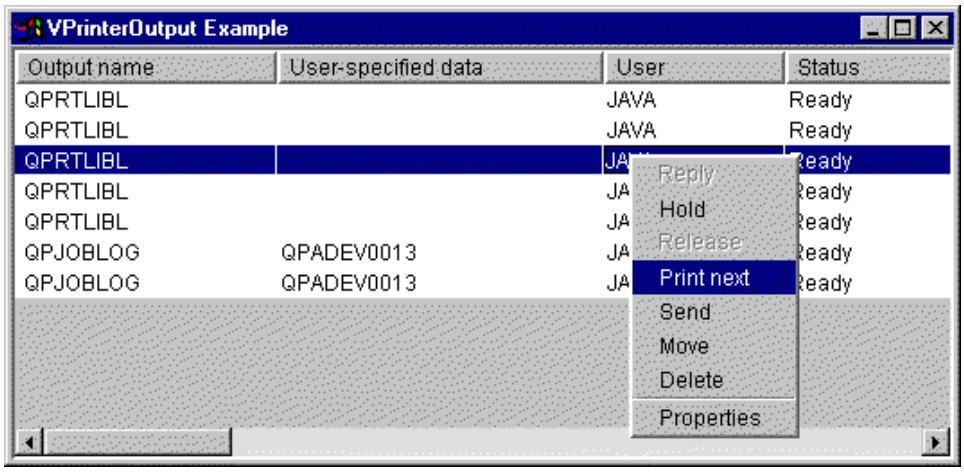
Il seguente esempio crea un VPrinterOutput e lo presenta in un AS400ListPane:

```
                                    // Creare l'oggetto VPrinterOutput.
                  // Supponiamo che "system" sia un oggetto AS400
                  // creato ed inizializzato
                  // da qualche altra parte.
VPrinterOutput root = new VPrinterOutput (system);

                                    // Creare e caricare un oggetto
                                    // AS400TreePane.
AS400ListPane listPane = new AS400ListPane (root);
       listPane.load ();

                                    // Aggiungere il pannello elenco ad una frame.
                                    // Supponiamo che "frame" sia un JFrame
                  // creato da qualche altra parte.
frame.getContentPane ().add (listPane);
```

Esempio

Presentare un elenco di file di spool utilizzando la risorsa di stampa, l'oggetto VPrinterOutput. La figura 1 illustra il componente GUI VPrinterOutput:

Figura 1: componente GUI VPrinterOutput



Classe SpooledFileViewer:

La classe SpooledFileViewer crea una finestra per la visualizzazione dei file AFP (Advanced Function Printing) e stringa di carattere SCS (Systems Network Architecture) che sono stati sottoposti a spool per la stampa.La classe aggiunge in sostanza una funzione di "anteprima di stampa" sui file sottoposti a spool, comune alla maggior parte di programmi di elaborazione, così come illustrato nella Figura 1.

Il programma di visualizzazione file di spool è di aiuto quando la visualizzazione della precisione della disposizione dei file è più importante della stampa dei file, quando la visualizzazione dei dati è più economica della stampa o quando non è disponibile una stampante.

Nota: sul server host deve essere installato SS1 Option 8 (AFP Compatibility Fonts).

Utilizzo della classe SpooledFileViewer

Sono disponibili tre metodi del programma di creazione per creare un'istanza della classe SpooledFileViewer. Il programma di creazione SpooledFileViewer() può essere utilizzato per creare un programma di visualizzazione senza file di spool ad esso associati.Se viene utilizzato questo programma di creazione, un file di spool dovrà essere impostato successivamente utilizzando setSpooledFile(SpooledFile).Il programma di creazione di SpooledFileViewer(SpooledFile) può essere utilizzato per creare un programma di visualizzazione per un determinato file di spool, con la prima pagina impostata come visualizzazione iniziale.Infine, il programma di creazione SpooledFileViewer(spooledFile, int) può essere utilizzato per creare un programma di visualizzazione per il file di spool con la pagina specificata impostata come visualizzazione iniziale.Una volta che è stato creato un programma di visualizzazione, non ha importanza quale programma di creazione si utilizza, deve essere eseguita una chiamata a load() per richiamare realmente i dati del file di spool.

Il programma può percorrere le singole pagine del file di spool utilizzando i metodi che seguono:

- load FlashPage()
- load Page()
- pageBack()
- pageForward()

Tuttavia, se l'utente ha bisogno di esaminare più approfonditamente sezioni particolari del documento, è possibile ingrandire o ridurre l'immagine di una pagina del documento alterando le proporzioni del rapporto di ogni pagina con quanto segue:

- fitHeight()
- fitPage()
- fitWidth()
- actualSize()

Il programma si conclude con la chiamata al metodo close() che chiude il flusso di immissione e rilascia qualsiasi associazione risorsa con il flusso.

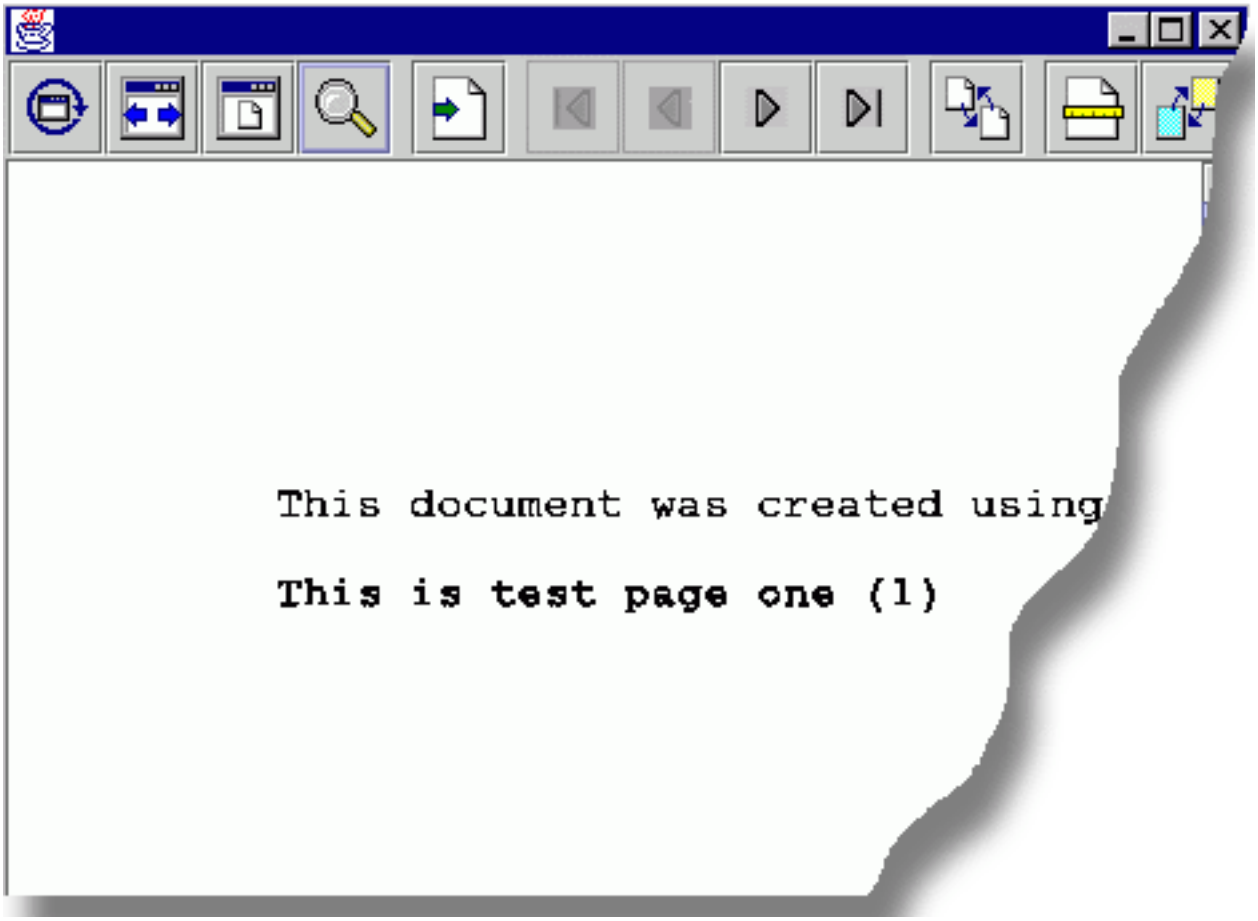
Utilizzo di SpooledFileViewer

Un'istanza della classe SpooledFileViewer è in effetti una rappresentazione grafica di un programma di visualizzazione capace di visualizzare ed esaminare il contenuto di un file di spool AFP o SCS. Ad esempio, il codice che segue crea il programma di visualizzazione file di spool nella Figura 1 per visualizzare un file di spool creato in precedenza sul server.

Nota: è possibile selezionare un pulsante sull'immagine nella Figura 1 per una spiegazione della relativa funzionalità o (se il browser non è abilitato JavaScript) consultare la descrizione della barra degli strumenti.

```
// Presupporre che splf sia il file di spool.
// Creare il programma di visualizzazione del file di spool
SpooledFileViewer splfv = new SpooledFileViewer(splf, 1);
splfv.load();
// Aggiungere il programma di visualizzazione del file di spool ad una frame
JFrame frame = new JFrame("My Window");
frame.getContentPane().add(splfv);
frame.pack();
         frame.show ();
```

Figura 1: SpooledFileViewer



Descrizione Barra degli strumenti SpooledFileViewer

 Il pulsante dimensione reale riporta l'immagine della pagina del file di spool alla dimensione originale utilizzando il metodo actualSize().

 Il pulsante adatta ampiezza allarga l'immagine della pagina del file di spool ai margini sinistro e destro della frame del programma di visualizzazione utilizzando il metodo fitWidth().

 Il pulsante adatta pagina amplia verticalmente e orizzontalmente l'immagine della pagina del file di spool in modo che si adatti alla frame del programma di visualizzazione del file di spool utilizzando il metodo fitPage().

 Il pulsante zoom consente di ingrandire o ridurre la dimensione dell'immagine della pagina del file di spool selezionando una delle percentuali preimpostate o immettendo la propria percentuale in un campo di testo che appare in una casella di dialogo dopo la selezione del pulsante zoom.

 Il pulsante vai a pagina consente di andare ad una pagina specifica entro il file di spool quando viene selezionato.

 Il pulsante prima pagina porta l'utente alla prima pagina del file di spool quando viene selezionato ed indica che l'utente si trova alla prima pagina quando è disattivato.

 Il pulsante pagina precedente porta l'utente alla pagina immediatamente precedente a quella che si sta visualizzando quando viene selezionato.

 Il pulsante pagina successiva fa avanzare l'utente alla pagina immediatamente successiva a quella che si sta visualizzando quando viene selezionato.

 Il pulsante ultima pagina fa avanzare l'utente fino all'ultima pagina del file di spool quando vene selezionato ed indica che l'utente si trova sull'ultima pagina quando è disattivato.

 Il pulsante carica pagina flash carica la pagina precedentemente visualizzata tramite il metodo loadFlashPage() quando viene selezionato.

 Il pulsante imposta dimensione foglio consente di impostare la dimensione del foglio quando viene selezionato.

 Il pulsante imposta risoluzione di visualizzazione consente di impostare la risoluzione della visualizzazione quando viene selezionato.

Classi Vaccess ProgramCall

I componenti di chiamata al programma nel pacchetto vaccess consentono ad un programma Java di presentare un pulsante o una voce del menu che richiama un programma del server. E' possibile specificare i parametri di immissione, emissione e immissione/emissione utilizzando gli oggetti ProgramParameter.Quando il programma è in esecuzione, i parametri di emissione e di immissione/emissione contengono i dati restituiti dal programma del server.

Un oggetto ProgramCallButton rappresenta un pulsante che richiama un programma del server quando viene premuto.La classe ProgramCallButton estende la classe JButton di JFC (Java Foundation Classes) in modo tale che tutti i pulsanti abbiano un aspetto e una funzionalità coerenti.

Ugualmente, un oggetto ProgramCallMenuItem rappresenta una voce del menu che richiama un programma del server quando viene selezionato.La classe ProgramCallMenuItem estende la classe JMenuItem JFC in modo tale che tutte le voci di menu abbiano un aspetto e una funzionalità coerenti.

Per utilizzare il componente di chiamata del programma vaccess, impostare sia le proprietà del programma che quelle di sistema. Impostare queste proprietà utilizzando un programma di creazione o attraverso i metodi setSystem() e setProgram().

Il seguente esempio crea un ProgramCallMenuItem. Nel tempo di esecuzione, quando si seleziona la voce del menu, richiama un programma:

```
                                        // Creare l'oggetto ProgramCallMenuItem.
                                        // AS400TreePane.
Supponiamo che "system" sia un oggetto
                                        // AS400 creato e inizializzato da qualche
                                        // altra parte.
Il testo
                    // della voce menu dice "Select Me" e
                                        // non vi è icona.
    ProgramCallMenuItem menuItem = new ProgramCallMenuItem ("Select Me", null, system);

                                        // Creare un oggetto nome percorso che
                                        // rappresenti il programma MYPROG nella
                                        // libreria MYLIB
    QSYSObjectPathName programName = new QSYSObjectPathName("MYLIB", "MYPROG", "PGM");

                                        // Impostare il nome del programma.
        menuItem.setProgram (programName.getPath());

                                        // Aggiungere la voce di menu ad un menu.
                                        // Supponiamo che il menu sia stato creato
                         // da qualche altra parte.
        menu.add (menuItem);
```

Quando si esegue un programma del server, è possibile che vengano restituito zero o più messaggi del server. Per rilevare quando il programma del server è in esecuzione, aggiungere un ActionCompletedListener al pulsante o alla voce del menu utilizzando il metodo addActionCompletedListener().Quando il programma è in esecuzione, invia un ActionCompletedEvent a tutti i listener.Un listener può utilizzare il metodo getMessageList() per richiamare qualsiasi messaggio del server creato dal programma.

Questo esempio aggiunge un ActionCompletedListener che elabora tutti i messaggi server che il programma ha generato:

```
                                        // Aggiungere un ActionCompletedListener
                                        // che sia implementato utilizzando una
                                        // classe interna anonima. Questo è un modo
                    // conveniente di specificare semplici
                                        // listener di eventi.
              menuItem.addActionCompletedListener (new ActionCompletedListener ()
    {
                        public void actionCompleted (ActionCompletedEvent event)
        {
                                        // Inserire il sorgente dell'evento in un
                                        // ProgramCallMenuItem.
                            ProgramCallMenuItem sourceMenuItem = (ProgramCallMenuItem) event.getSource ();

                    // Richiamare l'elenco dei messaggi server
                    // generata dal programma.
                AS400Message[] messageList = sourceMenuItem.getMessageList ();

                    // ... Elaborare l'elenco dei messaggi.
        }
    });
```

Parametri

Gli oggetti ProgramParameter vengono utilizzati per inviare dati di parametro tra il programma Java e il programma del server.I dati di immissione sono impostati con il metodo setInputData().Dopo l'esecuzione del programma, i dati di emissione sono richiamati con il metodo getOutputData().

Ogni parametro consiste in una schiera di byte. E' compito del programma Java convertire la schiera byte tra i formati Java e del server. Le classi di conversione dati forniscono metodi per la conversione dei dati.

E' possibile aggiungere dei parametri al componente GUI di chiamata al programma uno alla volta utilizzando il metodo addParameter() o contemporaneamente utilizzando il metodo setParameterList().

Per ulteriori informazioni sull'utilizzo degli oggetti ProgramParameter, vedere la classe di accesso ProgramCall.

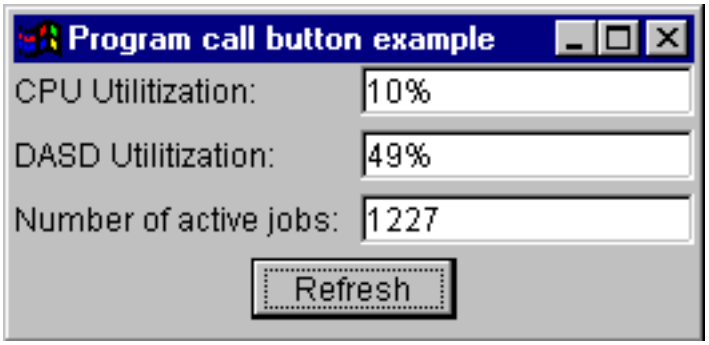
Il seguente esempio aggiunge due parametri:

```
                                    // Il primo parametro è un nome
                                    // Stringa lungo fino a 100 caratteri.
                                    // Questo è un parametro di immissione.
            // Supponiamo che "name" sia una Stringa
                                    // creata ed inizializzata altrove.
AS400Text parm1Converter = new AS400Text (100, system.getCcsid (), system);
ProgramParameter parm1 = new ProgramParameter (parm1Converter.toBytes (name));
menuItem.addParameter (parm1);

                                    // Il secondo parametro è un parametro
                                    // di emissione Numero intero.
      AS400Bin4 parm2Converter = new AS400Bin4 ();
ProgramParameter parm2 = new ProgramParameter (parm2Converter.getByteLength ());
menuItem.addParameter (parm2);

                                    // ... una volta chiamato il programma,
                                    // richiamare il valore restituito come
                                    // secondo parametro.
int result = parm2Converter.toInt (parm2.getOutputData ());
```

Esempi

Esempio di utilizzo di un ProgramCallButton in un'applicazione.La figura 1 illustra l'aspetto del ProgramCallButton:

Figura 2: utilizzo di ProgramCallButton in un'applicazione



Classi di accesso al livello record Vaccess

Le classi di accesso a livello record nel pacchetto vaccess consentono ad un programma Java di presentare diverse viste dei file del server.

I seguenti componenti sono disponibili:

- RecordListFormPane presenta un elenco di record che derivano da un file del server in un modulo.
- RecordListTablePane presenta un elenco di record che derivano da un un file del server in una tabella.
- RecordListTableModel gestisce l'elenco dei record che derivano da un file del server per una tabella.

Accesso con chiave

E' possibile utilizzare i componenti GUI di accesso a livello record con l'accesso con chiave ad un file del server. L'accesso con chiave significa che il programma Java può accedere ai record di un file specificando una chiave.

L'accesso con chiave funziona allo stesso modo per ogni componente GUI di accesso a livello record. Utilizzare setKeyed() per specificare l'accesso con chiave invece dell'accesso sequenziale. Specificare una chiave utilizzando il programma di creazione o il metodo setKey(). Consultare Specifica della chiave per ulteriori informazioni su come specificare la chiave.

Per impostazione predefinita, vengono visualizzati solo i record le cui chiavi sono uguali alla chiave specificata. Per modificare ciò, specificare la proprietà searchType utilizzando il programma di creazione o il metodo setSearchType(). Le possibili scelte sono:
- KEY_EQ - Visualizzare record le cui chiavi sono uguali alla chiave specificata.
- KEY_GE - Visualizzare i record le cui chiavi sono maggiori o uguali alla chiave specificata.
- KEY_GT - Visualizzare i record le cui chiavi sono maggiori della chiave specificata.
- KEY_LE - Visualizzare i record le cui chiavi sono minori o uguali alla chiave specificata.
- KEY_LT - Visualizzare i record le cui chiavi sono minori della chiave specificata.

Il seguente esempio crea un oggetto RecordListTablePane per visualizzare tutti i record minori o uguali alla chiave.

```
                                        // Creare una chiave che contiene un
                                        // singolo elemento, il Numero intero 5.
          Object[] key = new Object[1];
         key[0] = new Integer (5);

                                        // Creare un oggetto RecordListTablePane.
                                        // AS400TreePane.
Supponiamo che "system" sia un
                                        // oggetto AS400 creato e
                                        // altra parte.
Specificare
                                        // la chiave ed il tipo di ricerca.
         RecordListTablePane tablePane = new RecordListTablePane (system,
         "/QSYS.LIB/QGPL.LIB/PARTS.FILE", key, RecordListTablePane.KEY_LE);

                                        // Caricare il contenuto file.
                    tablePane.load ();

                                        // Aggiungere il pannello tabella ad una frame.
                                        // Supponiamo che "frame" sia un JFrame
                      // creato da qualche altra parte.
         frame.getContentPane ().add (tablePane);
```

Classe RecordListFormPane:

Un RecordListFormPane presenta il contenuto di un file del server in un modulo.Il modulo visualizza un record alla volta e fornisce i pulsanti che consentono all'utente di scorrere avanti, indietro, al primo o all'ultimo record o di aggiornare la vista del contenuto del file.

Per utilizzare un RecordListFormPane, impostare il sistema e le proprietà fileName. Impostare queste proprietà utilizzando il programma di creazione o setSystem() e i metodi setFileName().Utilizzare load()

per richiamare il contenuto del file e presentare il primo record.Quando il contenuto non è più necessario, richiamare close() per assicurarsi che il file sia chiuso.

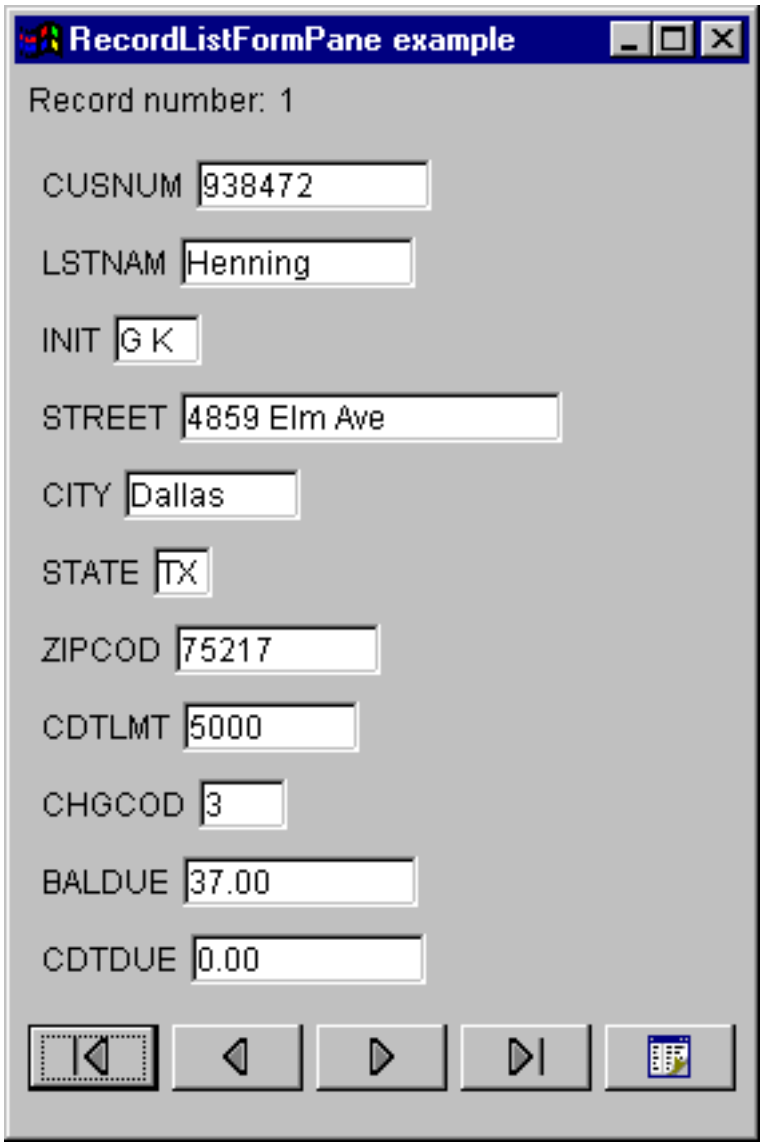
Il seguente esempio crea un oggetto RecordListFormPane e lo aggiunge ad una frame:

```
                                          // Creare un oggetto RecordListFormPane.
                                          // AS400TreePane.
Supponiamo che "system" sia un oggetto
                                          // AS400 creato ed
                                          // inizializzato altrove.
     RecordListFormPane formPane = new RecordListFormPane (system,
                                  "/QSYS.LIB/QIWS.LIB/QCUSTCDT.FILE");

                                          // Caricare il contenuto file.
           formPane.load ();

                                          // Aggiungere il pannello modulo ad una frame.
                                          // Supponiamo che "frame" sia un JFrame
                         // creato da qualche altra parte.
         frame.getContentPane ().add (formPane);
```

Esempio

Presentare un RecordListFormPane che visualizza il contenuto di un file.La figura 1 illustra il componente GUI di RecordListFormPane:

Figura 1: componente GUI RecordListFormPane



Classe RecordListTablePane:

Un RecordListTablePane presenta il contenuto di un file del server in una tabella.Nella tabella ogni riga visualizza un record del file e ogni colonna visualizza un campo.

Per utilizzare un RecordListTablePane, impostare le proprietà fileName e di sistema. Impostare queste proprietà utilizzando il programma di creazione o setSystem() e i metodi setFileName().Utilizzare load() per richiamare il contenuto del file e presentare i record nella tabella.Quando il contenuto non è più necessario, richiamare close() per assicurarsi che il file sia chiuso.

Il seguente esempio crea un oggetto RecordListTablePane e lo aggiunge ad una frame:

```
                                        // Creare un oggetto RecordListTablePane.
                                        // AS400TreePane.
Supponiamo che "system" sia un oggetto
                                        // AS400 creato ed
                                        // inizializzato altrove.
     RecordListTablePane tablePane = new RecordListTablePane (system,
                                    "/QSYS.LIB/QIWS.LIB/QCUSTCDT.FILE");
                                        // Caricare il contenuto file.
                         tablePane.load ();
```

```
                                        // Aggiungere il pannello tabella ad una frame.
                                        // Supponiamo che "frame" sia un JFrame
                    // creato da qualche altra parte.
        frame.getContentPane ().add (tablePane);
```

Classi RecordListTablePane e RecordListTableModel:

RecordListTablePane viene implementata utilizzando il paradigma programma di controllo vista modello, nel quale i dati e l'interfaccia utente sono separati in classi diverse. L'implementazione integra RecordListTableModel con JTable di JFC (Java Foundation Classes).La classe RecordListTableModel richiama e gestisce il contenuto del file e JTable visualizza graficamente il contenuto del file e gestisce l'interazione utente.

RecordListTablePane fornisce sufficiente funzionalità per la maggior parte dei requisiti. Tuttavia, se il programma di chiamata richiede un ulteriore controllo del componente JFC, può utilizzare direttamente RecordListTableModel e fornire un'integrazione personalizzata con un diverso componente GUI.

Per utilizzare un RecordListTableModel, impostare il sistema e le proprietà fileName. Impostare queste proprietà utilizzando il programma di creazione o setSystem() e i metodi setFileName().Utilizzare load() per richiamare il contenuto del file.Quando il contenuto non è più necessario, richiamare close() per assicurarsi che il file sia chiuso.

Il seguente esempio crea un oggetto RecordListTableModel e lo presenta con un JTable:

```
                                // Creare un oggetto RecordListTableModel.
                                        // AS400TreePane.
Supponiamo che "system" sia un oggetto
                                        // AS400 creato ed
                                        // inizializzato altrove.
     RecordListTableModel tableModel = new RecordListTableModel (system, "/QSYS.LIB/QIWS.LIB/QCUSTCDT.FILE");

                                        // Caricare il contenuto file.
          tableModel.load ();

                     // Creare una JTable per il modello.
          JTable table = new JTable (tableModel);

                     // Aggiungere la tabella ad una frame. Supponiamo
                     // che tale "frame" sia un JFrame
                     // creato da qualche altra parte.
          frame.getContentPane ().add (table);
```

ResourceListPane e ResourceListDetailsPane

Utilizzare le classi ResourceListPane e ResourceListDetailsPane per presentare un elenco di risorse in una GUI.

- ResourceListPane visualizza il contenuto di un elenco delle risorse in un javax.swing.JList grafico. Ogni voce visualizzata nell'elenco rappresenta un oggetto risorsa dall'elenco delle risorse.
- ResourceListDetailsPane visualizza il contenuto dell'elenco delle risorse in un javax.swing.JTable grafico. Ogni riga della tabella rappresenta un oggetto risorsa dall'elenco delle risorse.

Le colonne della tabella per un ResourceListDetailsPane sono specificate come una schiera di Id attributo della colonna. La tabella contiene una colonna per ogni elemento della schiera e una riga per ogni oggetto risorsa.

I menu a comparsa sono abilitati per impostazione predefinita sia per ResourceListPane che per ResourceListDetailsPane.

La maggior parte degli errori vengono riportati come com.ibm.as400.vaccess.ErrorEvents piuttosto che lanciati come eccezioni. Consultare ErrorEvents per effettuare una diagnosi e per correggere le condizioni di errore.

Esempio: visualizzazione di un elenco risorse in una GUI

Questo esempio crea un ResourceList di tutti gli utenti sul sistema e lo visualizza in una GUI (pannello dettagli):

```
    // Creare l'elenco risorse.
    AS400 system = new AS400("MYSYSTEM", "MYUSERID", "MYPASSWORD");
    RUserList userList = new RUserList(system);

    // Creare ResourceListDetailsPane.  In questo esempio,
    // vi sono due colonne nella tabella.  La prima colonna
    // contiene le icone ed i nomi per ogni utente. Le
    // seconda colonna contiene la descrizione testo per ogni
    // utente.
    Object[] columnAttributeIDs = new Object[] { null, RUser.TEXT_DESCRIPTION };
    ResourceListDetailsPane detailsPane = new ResourceListDetailsPane();
    detailsPane.setResourceList(userList);
    detailsPane.setColumnAttributeIDs(columnAttributeIDs);

    // Aggiungere il ResourceListDetailsPane ad un JFrame e visualizzarlo.
JFrame frame = new JFrame("My Window");
    frame.getContentPane().add(detailsPane);
frame.pack();
frame.show();

    // Il ResourceListDetailsPane apparirà vuoto fino a quando
    // non viene caricato.  Questo permette di controllare quando l'elenco
     // di utenti viene richiamato dall'iSeries.
            detailsPane.load ();
```

Classi SystemStatus

I componenti System Status nel pacchetto vaccess consentono di creare le GUI utilizzando AS400Panes esistenti.Inoltre è disponibile l'opzione che consente di creare GUI proprie utilizzando le JFC (Java Foundation Classes). L'oggettoVSystemStatus rappresenta lo stato del sistema sul server.L'oggetto VSystemPool rappresenta un lotto di sistema sul server.VSystemStatusPane rappresenta un pannello visivo che visualizza le informazioni sullo stato del sistema.

La classe VSystemStatus consente di richiamare le informazioni sullo stato di una sessione server nell'ambiente GUI:

- Il metodo getSystem() indica il server che contiene le informazioni sul lotto di sistema
- Il metodo getText() restituisce il testo della descrizione
- Il metodo setSystem() imposta il server che contiene le informazioni sullo stato del sistema

Oltre ai metodi menzionati in precedenza, è possibile anche accedere e modificare le informazioni sul lotto di sistema in una GUI.

Utilizzare VSystemStatus con VSystemStatusPane.VSystemPane è il pannello visivo su cui vengono visualizzate le informazioni sullo stato del sistema e sul lotto di sistema.

Classe VSystemPool:

La classe VSystemPool consente di richiamare e impostare le informazioni sul lotto di sistema da un server, utilizzando un progetto GUI.VSystemPool utilizza diversi pannelli nel pacchetto vaccess, incluso VSystemStatusPane.

L'elenco seguente contiene alcuni metodi disponibili da utilizzare in VSystemPool:

- Il metodo getActions() restituisce un elenco di operazioni da eseguire
- Il metodo getSystem() restituisce il server in cui è possibile reperire le informazioni sul lotto di sistemi

- Il metodo setSystemPool() imposta l'oggetto del lotto di sistema

Classe VSystemStatusPane:

La classe VSystemStatusPane consente ad un programma Java di visualizzare lo stato del sistema e le informazioni sul lotto di sistema.

VSystemStatusPane comprende i seguenti metodi:
- getVSystemStatus(): restituisce le informazioni VSystemStatus in un VSystemStatusPane.
- setAllowModifyAllPools(): imposta il valore per stabilire se le informazioni sul lotto di sistema possono essere modificate.

Il seguente esempio visualizza come utilizzare la classe VSystemStatusPane:

```
// Creare un oggetto as400.
AS400 mySystem = new AS400("mySystem.myCompany.com");

// Creare un VSystemStatusPane
VSystemStatusPane myPane = new VSystemStatusPane(mySystem);

// Impostare il valore per consentire la modifica dei lotti
myPane.setAllowModifyAllPools(true);

// Caricare le informazioni
myPane.load();
```

GUI di SystemValues

I componenti SystemValues nel pacchetto vaccess consentono a un programma Java di creare le GUI utilizzando AS400Panes esistenti o creando pannelli propri tramite le JFC (Java Foundation Classes).L'oggetto VSystemValueList rappresenta un elenco dei valori di sistema nel server.

Per utilizzare il componente GUI dei valori di sistema, impostare il nome del sistema con un programma di creazione o con il metodo setSystem().

Esempio Il seguente esempio crea una GUI di valori di sistema che utilizzano l'AS400Explorer Pane:

```
//Creare un oggetto AS400
AS400 mySystem = newAS400("mySystem.myCompany.com");
VSystemValueList mySystemValueList = new VSystemValueList(mySystem);
as400Panel=new AS400ExplorerPane((VNode)mySystemValueList);
//Creare e caricare un oggetto AS400ExplorerPane
as400Panel.load();
```

Classi Users e Groups Vaccess

I componenti Users e Groups nel pacchetto vaccess consentono di presentare l'elenco degli utenti del server e dei gruppi attraverso la Classe VUser.

I seguenti componenti sono disponibili:
- AS400Panes sono componenti GUI che presentano e consentono la gestione di una o più risorse del server.
- Un oggetto VUserList è una risorsa che rappresenta un elenco di utenti e di gruppi del server da utilizzare in AS400Panes.
- Un oggetto VUserAndGroup è una risorsa da utilizzare in AS400Panes che rappresenta i gruppi degli utenti del server. Consente ad un programma Java di elencare tutti gli utenti, tutti i gruppi o gli utenti che non fanno parte dei gruppi.

E' possibile utilizzare insieme gli oggetti AS400Pane e VUserList per presentare diverse viste dell'elenco. Possono anche essere utilizzati per consentire all'utente di selezionare gli utenti e i gruppi.

Per utilizzare un VUserList, è necessario innanzitutto impostare la proprietà di sistema. Impostare questa proprietà utilizzando un programma di creazione o il metodo setSystem(). L'oggetto VUserList viene quindi collegato all'AS400Pane come root, utilizzando il programma di creazione o il metodo setRoot() di AS400Pane.

VUserList ha altre proprietà utili per definire la serie di utenti e di gruppi che sono presentati in AS400Panes:

- Utilizzare il metodo setUserInfo() per specificare i tipi di utenti che dovrebbero essere visualizzati.
- Utilizzare il metodo setGroupInfo() per specificare un nome del gruppo.

E' possibile utilizzare VUserAndGroup per ricevere le informazioni su Utenti e Gruppi sul sistema.Prima di poter ricevere le informazioni su un oggetto particolare, è necessario caricare le informazioni in modo tale che sia possibile accedervi.E' possibile visualizzare il server in cui si trovano le informazioni utilizzando il metodo getSystem.

Quando vengono creati gli oggetti AS400Pane e VUserList o VUserAndGroup, sono inizializzati su uno stato predefinito. L'elenco degli utenti e dei gruppi non è stato caricato. Per caricare il contenuto, è necessario che il programma Java richiami esplicitamente il metodo load() su uno dei due oggetti per iniziare la comunicazione con il server al fine di raccogliere il contenuto dell'elenco.

Al momento dell'esecuzione, fare clic con il tastino destro del mouse su un utente, un elenco utenti o sui gruppi per visualizzare il menu di scelta rapida. Selezionare **Proprietà** dal menu di scelta rapida per eseguire le azioni sull'oggetto selezionato:

- Utente - Visualizzare un elenco di informazioni sull'utente comprese la descrizione, la classe dell'utente, lo stato, la descrizione del lavoro, le informazioni sull'emissione, le informazioni sul messaggio, le informazioni internazionali, le informazioni sulla sicurezza e le informazioni sul gruppo.
- Elenco utenti - Gestire le proprietà delle informazioni sull'utente e sul gruppo. E' inoltre possibile modificare il contenuto dell'elenco.
- Utenti e gruppi - Visualizzare le proprietà, come il nome utente e la descrizione.

Gli utenti possono accedere solo agli utenti e ai gruppi ai quali sono autorizzati. Inoltre, è possibile che il programma Java impedisca all'utente di eseguire le operazioni utilizzando il metodo setAllowActions() sul pannello.

Il seguente esempio crea un VUserList e lo presenta in un AS400DetailsPane:

```
// Creare l'oggetto VUserList.
// Supponiamo che "system" sia un oggetto AS400
// creato ed inizializzato
// da qualche altra parte.
VUserList root = new VUserList (system);

// Creare e caricare un
// oggetto AS400DetailsPane.
AS400DetailsPane detailsPane = new AS400DetailsPane (root);
        detailsPane.load ();

// Aggiungere il pannello dettagli ad una frame.
                    // Supponiamo che "frame" sia un JFrame
// creato da qualche altra parte.
frame.getContentPane().add(detailsPane);
```

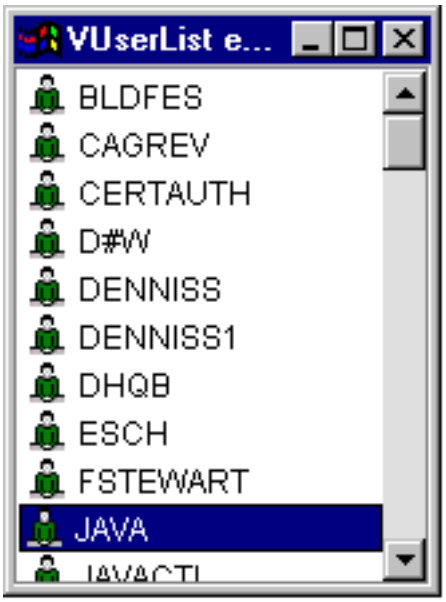
Il seguente esempio mostra come utilizzare l'oggetto VUserAndGroup:

```
// Creare l'oggetto VUserAndGroup.
// Supponiamo che "system" sia un oggetto AS400 creato ed inizializzato altrove.
VUserAndGroup root = new VUserAndGroup(system);

// Creare e caricare un AS400ExplorerPane
AS400ExplorerPane explorerPane = new AS400ExplorerPane(root);
        explorerPane.load ();

// Aggiungere il pannello explorer ad una frame
// Supponiamo che "frame" sia un JFrame creato altrove
frame.getContentPane().add(explorerPane);
```

Altri esempi

Presentare un elenco di utenti sul sistema utilizzando un AS400ListPane con un oggetto VUserList.

La seguente immagine visualizza il componente GUI di VUserList:



Graphical Toolbox

Il Graphical Toolbox, una serie di strumenti UI, facilita la creazione dei pannelli dell'interfaccia utente personalizzati in Java. E' possibile incorporare i pannelli nelle applicazioni Java, nelle applet o nei Moduli aggiuntivi di iSeries Navigator.I pannelli possono contenere dati derivati dall'iSeries o dati derivati da un altro sorgente, ad esempio da un file presente nel file system locale o da un programma nella rete.

Il **GUI Builder** è un editor visivo WYSIWYG per la creazione di finestre di dialogo, di fogli delle proprietà e di wizard Java.Con il GUI Builder è possibile aggiungere, disporre o modificare i controlli dell'interfaccia utente su un pannello e quindi visualizzare in anteprima il pannello per verificare che il layout si comporti nel modo previsto. E' possibile utilizzare le definizioni di pannello create nelle finestre di dialogo, inserirle nei fogli delle proprietà e nei wizard o disporle nei pannelli suddivisi, sovrapposti e con separatore. Il GUI Builder consente inoltre di creare barre dei menu, barre degli strumenti e definizioni dei menu di contesto. E' inoltre possibile incorporare JavaHelp nei pannelli, incluso l'aiuto sensibile al contesto.

Il **Resource Script Converter** converte gli script delle risorse Windows in una rappresentazione XML utilizzabile dai programmi Java.Con il Resource Script Converter è possibile elaborare gli script delle risorse di Windows (file RC) dalle finestre di dialogo e dai menu esistenti di Windows. E' quindi

possibile modificare questi file convertiti con il GUI Builder. E' possibile creare i fogli delle proprietà ed i wizard dai file RC utilizzando il Resource Script Converter insieme al GUI Builder.

Alla base di questi due strumenti vi è una nuova tecnologia denominata **Panel Definition Markup Language** o **PDML**.Il PDML è basato a sua volta sull'XML (Extensible Markup Language) e definisce un linguaggio indipendente dalla piattaforma per la descrizione del layout degli elementi dell'interfaccia utente. Dopo aver definito i pannelli in PDML, è possibile utilizzare l'API del tempo di esecuzione fornita dal Graphical Toolbox per visualizzarli. L'API visualizza i pannelli interpretando il PDML e restituendo l'interfaccia utente tramite le classi Foundation Java.

Nota: l'utilizzo di PDML richiede l'utilizzo della versione 1.4 o successiva di Java Runtime Environment.

Vantaggi del Graphical Toolbox

Scrivere una minor quantità di codice e risparmiare tempo
Con il Graphical Toolbox si ha la possibilità di creare velocemente e facilmente delle interfacce utente basate su Java. Il GUI Builder consente di effettuare un controllo capillare del layout degli elementi interfaccia utente (UI) nei pannelli. Dal momento che il layout viene descritto in PDML, non è necessario sviluppare alcun codice Java per definire l'interfaccia utente e non è necessario ricompilare il codice per effettuare delle modifiche. Di conseguenza, è necessaria una minore quantità di tempo per creare ed effettuare le applicazioni Java. Il Resource Script Converter consente di migrare velocemente e facilmente un notevole numero di pannelli Windows in Java.

Aiuto personalizzato
La definizione delle interfacce utente in PDML offre degli ulteriori vantaggi. Dal momento che tutte le informazioni relative ad un pannello vengono consolidate in un linguaggio con markup formale, è possibile potenziare gli strumenti per eseguire ulteriori servizi per conto dello sviluppatore. Ad esempio, sia il GUI Builder che il Resource Script Converter possono creare delle strutture HTML per l'aiuto in linea del pannello. Si decide quali argomenti di aiuto siano necessari e gli argomenti di aiuto vengono automaticamente creati in base ai requisiti richiesti. Le tag Ancora per gli argomenti di aiuto vengono create proprio nella struttura di aiuto, in modo da consentire allo sviluppatore dell'aiuto di concentrarsi sullo sviluppo del contenuto appropriato. L'ambiente del tempo di esecuzione del Graphical Toolbox visualizza automaticamente l'argomento di aiuto corretto in risposta alla richiesta dell'utente.

Pannello automatico per l'integrazione del codice
Inoltre, il PDML fornisce delle tag che associano ogni controllo in un pannello ad un attributo in un JavaBean. Dopo aver identificato le classi bean che forniranno dati al pannello ed aver associato un attributo ad ognuno degli controlli appropriati, è possibile richiedere che gli strumenti creino delle strutture di codice sorgente Java per gli oggetti bean. Durante il tempo di esecuzione, il Graphical Toolbox trasferisce automaticamente i dati fra i bean ed i controlli sul pannello identificato.

Indipendente dalla piattaforma
L'ambiente del tempo di esecuzione del Graphical Toolbox fornisce supporto per la gestione di eventi, per la convalida dei dati utente e per i comuni tipi di interazione fra gli elementi di un pannello. L'aspetto corretto della piattaforma per l'interfaccia utente viene automaticamente impostato in base al sistema operativo ed il GUI Builder consente di alternare l'aspetto per poter valutare il modo in cui verranno visualizzati i pannelli su diverse piattaforme.

Il Graphical Toolbox fornisce due strumenti e, quindi, due modi per automatizzare la creazione delle interfacce utente. E' possibile utilizzare il GUI Builder per creare da zero, in modo semplice e veloce, i nuovi pannelli oppure è possibile utilizzare il Resource Script Converter per convertire in Java i pannelli esistenti basati su Windows. I file convertiti possono essere quindi modificati con il GUI Builder. Entrambi gli strumenti supportano l'internazionalizzazione.

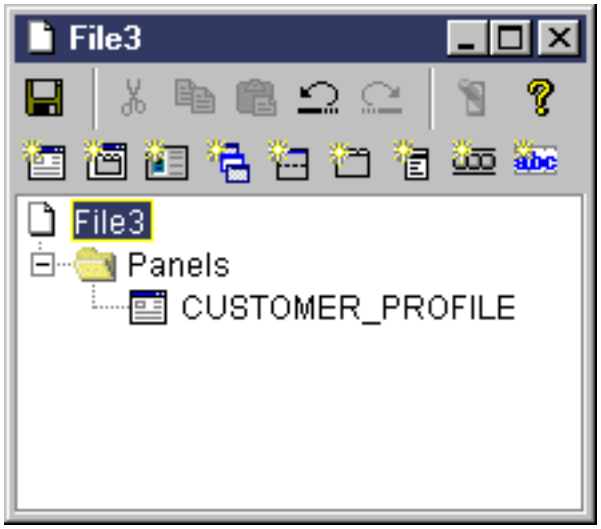
GUI Builder

Quando si richiama per la prima volta il GUI Builder, vengono visualizzate due finestre, come mostrato nella Figura 1:

Figura 1: finestre del GUI Builder

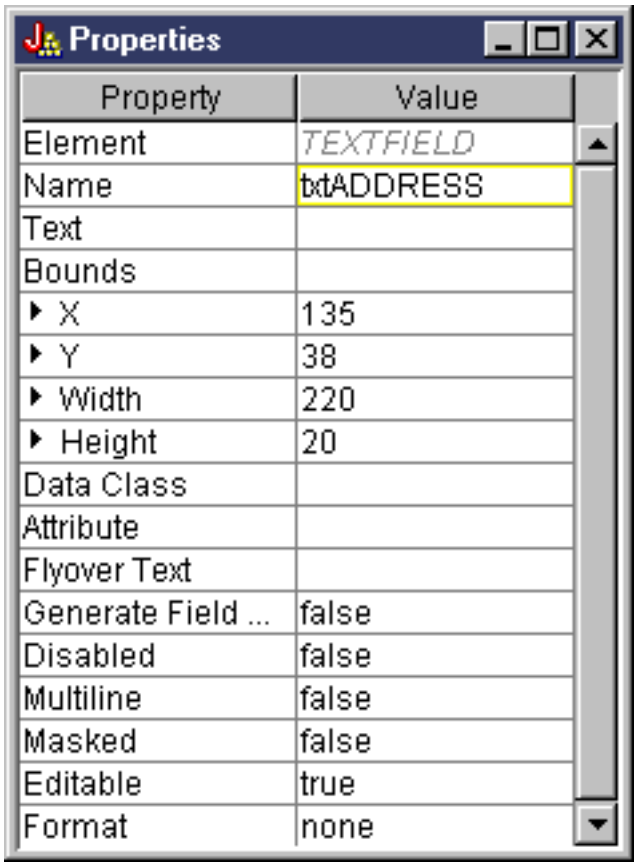
Utilizzare la finestra Programma di creazione del file per creare e modificare i file PDML.

Figura 2: finestra del programma di creazione del file



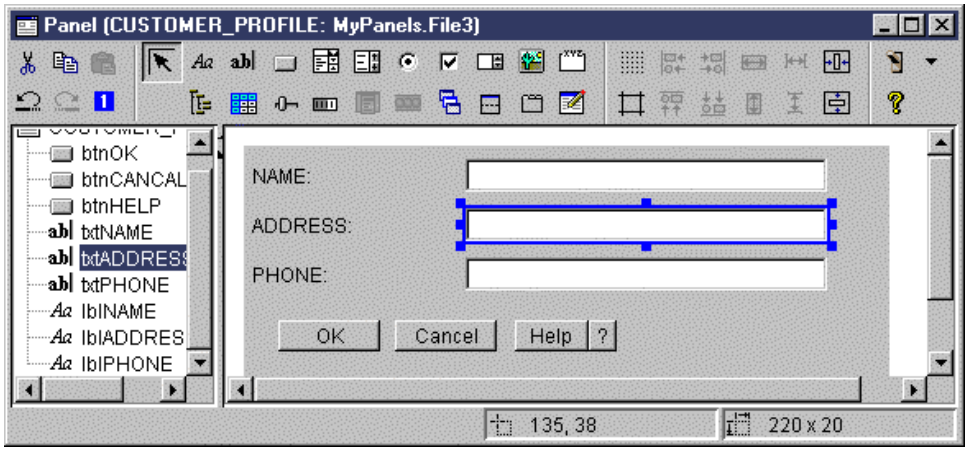
Utilizzare la finestra Proprietà per visualizzare o modificare le proprietà del controllo attualmente selezionato.

Figura 3: finestra Proprietà



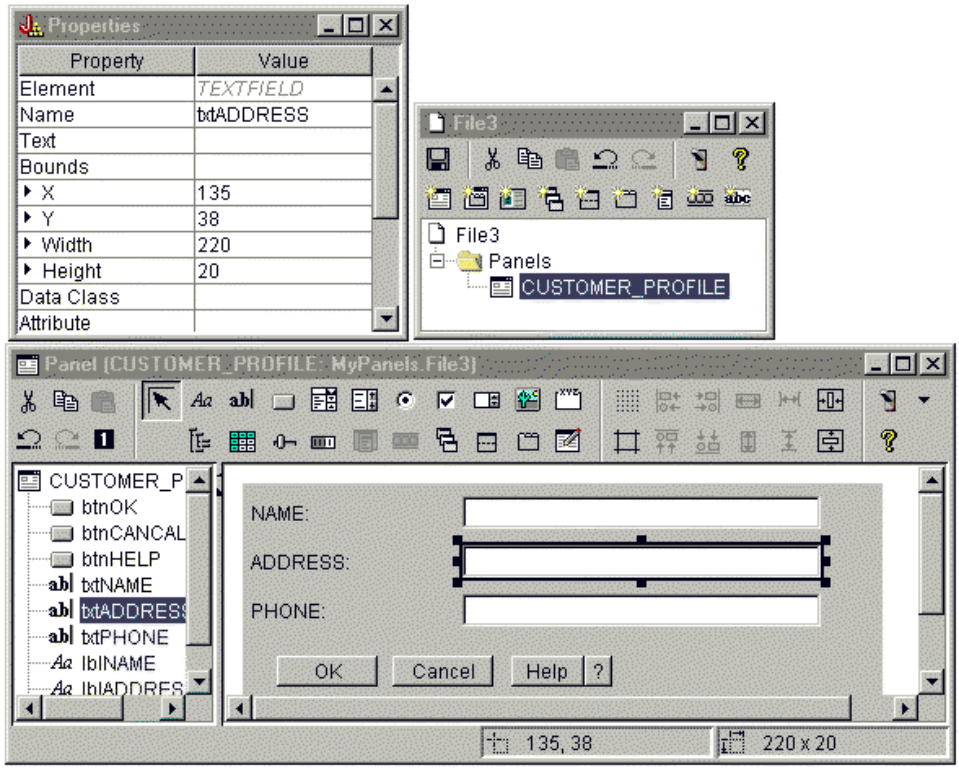
Utilizzare la finestra Programma di creazione del pannello per creare e modificare i componenti della GUI. Selezionare il componente desiderato dalla barra degli strumenti e fare clic sul pannello per collocarlo dove si desidera. La barra degli strumenti facilita inoltre l'allineamento dei gruppi dei controlli, la visualizzazione in anteprima del pannello e la richiesta dell'aiuto in linea per una funzione del GUI Builder. Consultare Barra degli strumenti del programma di creazione del pannello del GUI Builder per una descrizione delle funzioni di ogni icona.

Figura 4: finestra del programma di creazione del pannello



Il pannello modificato viene visualizzato nella finestra del programma di creazione del pannello. La figura 5 mostra il funzionamento delle finestre:

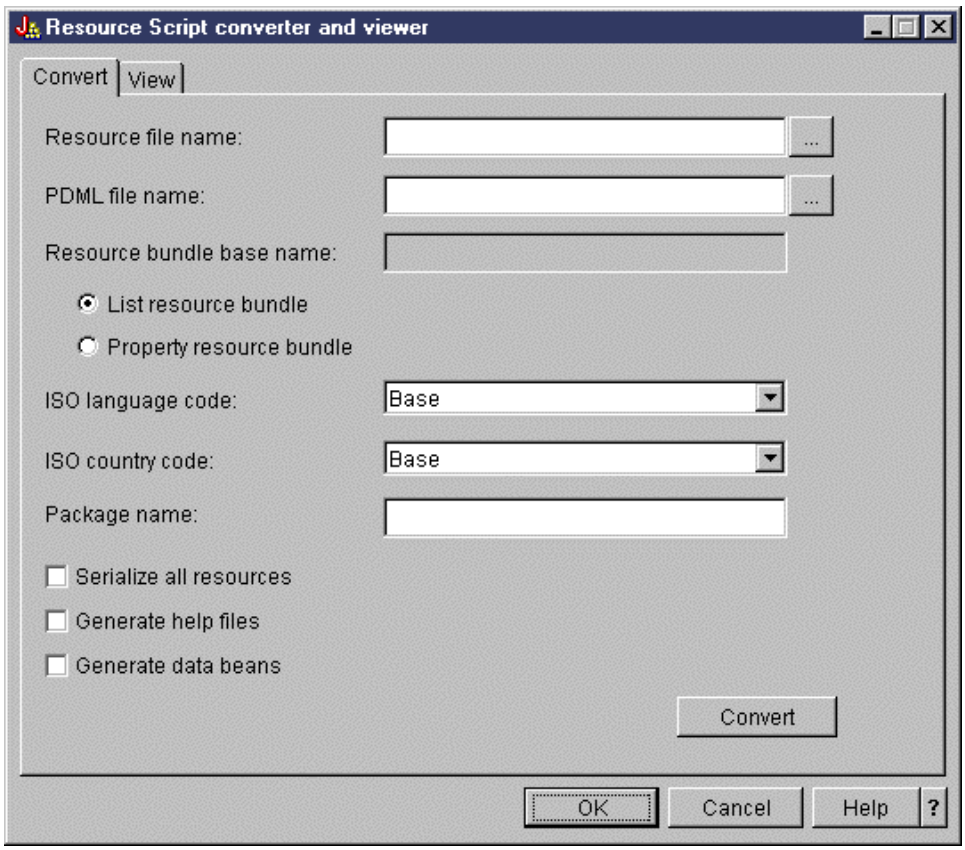
Figura 5: esempio di funzionamento congiunto delle finestre del GUI Builder



Resource Script Converter

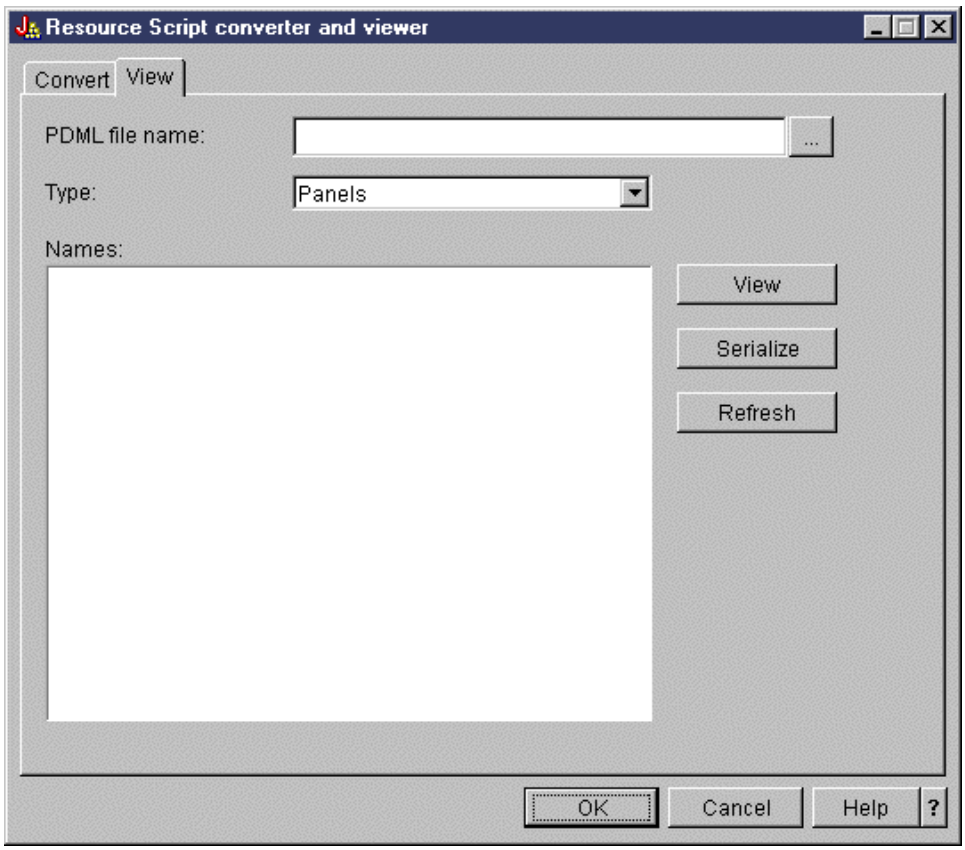
Il Resource Script Converter è costituito da una finestra di dialogo con separatori a due pannelli. Nel pannello **Conversione** si specifica il nome del file RC Microsoft o VisualAge per Windows che è necessario convertire in PDML. E' possibile specificare il nome del file PDML di destinazione e del bundle delle risorse Java associato che conterrà le stringhe convertite per i pannelli. E' inoltre possibile richiedere che vengano create le strutture dell'aiuto in linea per i pannelli, creare le strutture del codice sorgente Java per gli oggetti che forniscono i dati ai pannelli e serializzare le definizioni di pannello per incrementare le prestazioni durante il tempo di esecuzione. L'aiuto in linea del Converter fornisce una descrizione dettagliata di ogni campo di immissione nel pannello Conversione.

Figura 6: pannello Conversione Resource Script Converter



Dopo aver effettuato con esito positivo la conversione, è possibile utilizzare il Pannello **Visualizzazione** per visualizzare il contenuto del file PDML creato e per visualizzare in anteprima i nuovi pannelli Java. Se necessario, è possibile utilizzare il GUI Builder per effettuare piccoli adattamenti in un pannello. Il Converter verifica sempre l'esistenza di un file PDML prima di effettuare una conversione e tenta di conservare ogni modifica, nel caso sia successivamente necessario eseguire di nuovo la conversione.

Figura 7: pannello Visualizzazione Resource Script Converter



Informazioni preliminari su Graphical Toolbox

Utilizzare i seguenti argomenti per acquisire ulteriori nozioni sul Graphical Toolbox:

- Installazione del Graphical Toolbox
- Creazione dell'interfaccia utente
- Visualizzazione dei pannelli al tempo di esecuzione
- Creazione di file di aiuto in linea
- Esempio del Graphical Toolbox
- Utilizzo del Graphical Toolbox in un browser
- Barra degli strumenti del programma di creazione del pannello

Configurazione del Graphical Toolbox

Il Graphical Toolbox viene fornito come una serie di file JAR. Per installare il Graphical Toolbox è necessario installare i file JAR sulla propria stazione di lavoro ed impostare la variabile di ambiente CLASSPATH.

Inoltre ci si deve assicurare che la propria stazione di lavoro soddisfi i requisiti per l'esecuzione di IBM Toolbox per Java.

Installazione del Graphical Toolbox sulla stazione di lavoro

Per sviluppare programmi Java utilizzando il Graphical Toolbox, occorre innanzitutto installare i file JAR del Graphical Toolbox sulla stazione di lavoro. Utilizzare uno dei seguenti metodi:

Trasferimento dei file JAR

Nota: l'elenco che segue rappresenta alcuni dei metodi che possono essere utilizzati per trasferire

i file JAR. E' necessario installare sul proprio iSeries il programma su licenza IBM Toolbox per Java. Inoltre, è necessario scaricare il file JAR per JavaHelp, jhall.jar, dal sito web di Sun JavaHelp  .

- Utilizzare FTP (verificare che il trasferimento dei file avvenga in modalità binaria) e copiare i file JAR dall'indirizzario **/QIBM/ProdData/HTTP/Public/jt400/lib** su un indirizzario locale nella propria stazione di lavoro
- Utilizzare iSeries Access per Windows per definire una unità di rete.
- Si può inoltre utilizzare la classe AS400ToolboxInstaller fornita con IBM Toolbox per Java per installare i file JAR Graphical Toolbox - specificare OPNAV come nome del pacchetto. Per ulteriori informazioni, consultare Installazione del client e classi aggiornamento.

Installazione del file JAR con iSeries Access per Windows

E' inoltre possibile installare il Graphical Toolbox quando si installa iSeries Access per Windows. IBM Toolbox per Java viene ora fornito come parte di iSeries Access per Windows. Se si sta installando iSeries Access per Windows per la prima volta, scegliere Installazione personalizzata e selezionare il componente **IBM Toolbox per Java** sul menu di installazione. Se iSeries Access per Windows è già stato installato, è possibile utilizzare il programma di Installazione selective per installare questo componente se non è già presente.

Impostazione del classpath

Per utilizzare il Graphical Toolbox, è necessario aggiungere questi file JAR alla propria variabile d'ambiente CLASSPATH (o specificare tali file nell'opzione `percorso classe` sulla riga comandi).

Ad esempio, se i file sono stati copiati sull'indirizzario **C:\gtbox\lib** nella propria stazione di lavoro, è necessario aggiungere i seguenti nomi percorso al proprio classpath:

```
C:\gtbox\lib\uitools.jar;
     C:\gtbox\lib\jui400.jar;
     C:\gtbox\lib\data400.jar;
     C:\gtbox\lib\util400.jar;
     C:\gtbox\lib\jhall.jar;
```

E' inoltre necessario aggiungere il programma di analisi XML al CLASSPATH. Per ulteriori informazioni, consultare la seguente pagina:

"Programma di analisi XML e processore XSLT" a pagina 417

Se il Graphical Toolbox è stato installato utilizzando iSeries Access per Windows, i file JAR (eccetto jhall.jar) risiederanno tutti nell'indirizzario **\Program Files\Ibm\Client Access\jt400\lib** sull'unità in cui è stato installato iSeries Access per Windows.iSeries Access per Windows installa jhall.jar nell'indirizzario **\Program Files\Ibm\Client Access\jre\lib**.I nomi di percorso presenti nel classpath riportano quanto segue.

Descrizioni dei file JAR

- **uitools.jar**: contiene gli strumenti GUI Builder e Resource Script Converter.
- **jui400.jar**: contiene l'API del tempo di esecuzione per il Graphical Toolbox. I programmi Java utilizzano tale API per visualizzare i pannelli creati utilizzando gli strumenti. Tali classi possono essere ridistribuite con applicazioni.
- **data400.jar**: contiene l'API del tempo di esecuzione per il PCML. I programmi Java utilizzano questa API per chiamare i programmi iSeries i cui parametri e valori di restituzione vengono identificati utilizzando PCML. Queste classi possono essere ridistribuite con le applicazioni.
- **util400.jar**: contiene le classi Utility per la formattazione dei dati iSeries e la gestione dei messaggi iSeries. Tali classi possono essere ridistribuite con applicazioni.

- **jhall.jar**: contiene le classi JavaHelp che visualizzano l'aiuto in linea e l'aiuto sensibile al contesto per i pannelli creati con il GUI Builder.
- **XML parser**: contiene il programma di analisi XML utilizzato dalle classi API per interpretare i documenti PDML e PCML.

Nota: è possibile utilizzare le versioni internazionalizzate degli strumenti GUI Builder e Resource Script Converter. Per eseguire una versione in una lingua diversa dall'Inglese Americano, è necessario aggiungere alla propria installazione di Graphical Toolbox la versione corretta di **uitools.jar** per la lingua, il paese o la regione.Questi file JAR sono disponibili sul server iSeries in **/QIBM/ProdData/HTTP/Public/jt400/Mri29xx**, dove 29xx è il codice a 4 cifre della NLV OS/400, che corrisponde alla lingua, al paese o alla regione.(I nomi dei file JAR nei vari indirizzari Mri29xx includono suffissi di 2 caratteri per il codice lingua Java e il codice paese o regione.) Tale file JAR aggiuntivo deve essere aggiunto al classpath prima di **uitools.jar** nell'ordine di ricerca.

Utilizzo del Graphical Toolbox

Una volta che il Graphical Toolbox è stato installato, seguire questi collegamenti per acquisire informazioni su come utilizzare gli strumenti:

- Utilizzare GUI Builder
- Utilizzare Resource Script Converter

Creazione dell'interfaccia utente

Per avviare il GUI Builder, utilizzare il seguente comando:

```
java com.ibm.as400.ui.tools.GUI Builder [-plaf look and feel]
```

Se non è stata impostata la variabile d'ambiente CLASSPATH per contenere i file JAR del Graphical Toolbox, sarà necessario specificarli sulla riga comandi utilizzando l'opzione `classpath`. Consultare Installare il Graphical Toolbox.

Opzioni `-plaf` *look and feel*

L'aspetto desiderato della piattaforma. Questa opzione consente di sostituire l'aspetto predefinito impostato, in base alla piattaforma che si sta sviluppando, in questo modo è possibile visualizzare in anteprima i pannelli per verificare come appariranno su differenti piattaforme del sistema operativo. I seguenti valori relativi all'aspetto sono accettati:

- Windows
- Metal
- Motif

Attualmente, altri attributi di aspetto che Swing 1.1 può supportare non sono supportati dal GUI Builder

Tipi di risorse interfaccia utente

Quando il GUI Builder viene avviato per la prima volta, è necessario creare un nuovo file PDML. Dalla barra dei menu nella finestra GUI Builder, selezionare **File** --> **Nuovo File**. Dopo aver creato il nuovo file PDML, è possibile definire uno qualsiasi dei seguenti tipi di risorse UI che si desidera esso contenga.

Pannello

Il tipo di risorsa fondamentale. Descrive un'area rettangolare nella quale vengono disposti gli elementi UI. Gli elementi UI possono essere semplici controlli, quali i pallini o campi di testo, immagini, animazioni, controlli personalizzati o sottopannelli più sofisticati (consultare le seguenti definizioni per Pannello suddiviso, Pannello sovrapposto e Pannello con separatori). Un pannello può definire il layout per una finestra o una finestra di dialogo autonoma oppure può definire uno dei sottopannelli che sono contenuti in un'altra risorsa UI.

Menu Una finestra a comparsa che contiene una o più operazioni selezionabili, ognuna delle quali è rappresentata da una stringa di testo ("Taglia", "Copia" e "Incolla" sono esempi). E' possibile definire mnemonic e tasti di scelta rapida per ogni operazione, inserire separatori e sottomenu a cascata oppure definire voci di menu speciali selezionate tramite segno di spunta o pallino. Una risorsa di menu può essere utilizzata come menu contesto autonomo, come menu a discesa in una barra dei menu o può essa stessa definire la barra dei menu associata ad una risorsa pannello.

Barra degli strumenti
Una finestra che consiste in una serie di pulsanti, ognuno dei quali rappresenta una possibile operazione dell'utente. Ogni pulsante può contenere un testo, una icona o entrambi. E' possibile definire la barra degli strumenti come mobile, in modo che l'utente possa trascinarla al di fuori del pannello e in una finestra autonoma.

Foglio delle proprietà
Una finestra o una finestra di dialogo autonoma costituita da pannelli con separatori e pulsanti OK, Annulla e Aiuto. Le risorse pannello definiscono il layout di ogni finestra con separatori.

Wizard
Una finestra o una finestra di dialogo autonoma, costituita da una serie di pannelli visualizzati dall'utente in una sequenza predefinita, con pulsanti Indietro, Avanti, Annulla, Fine e Aiuto. La finestra wizard può anche visualizzare un elenco di attività a sinistra dei pannelli che tracciano l'avanzamento dell'utente nel wizard.

Pannello suddiviso
Un sottopannello composto da due pannelli separati da una barra di separazione. I pannelli possono essere disposti orizzontalmente o verticalmente.

Pannello con separatori
Un sottopannello che crea un controllo con separatori. Questo controllo con separatori può essere collocato all'interno di un altro pannello, in un pannello suddiviso o in un pannello sovrapposto.

Pannello sovrapposto
Un sottopannello costituito da una raccolta di pannelli. Di questi, può essere visualizzato un solo pannello alla volta. Ad esempio, in fase di esecuzione il pannello sovrapposto potrebbe modificare il pannello visualizzato in base ad una determinata operazione dell'utente.

Tabella stringa
Una raccolta di risorse stringa e gli identificativi di risorsa associate.

File creati

Le stringhe convertibili per un pannello non sono memorizzate nel file PDML stesso, ma in un bundle di risorse Java separato. Gli strumenti consentono di specificare come viene definito il bundle di risorse, come file PROPERTIES di Java o come sottoclasse ListResourceBundle. Una sottoclasse ListResourceBundle è una versione compilata delle risorse convertibili, che incrementa le prestazioni dell'applicazione Java. Tuttavia, rallenterà il processo di salvataggio del GUI Builder, poiché ListResourceBundle verrà compilata in ogni operazione di salvataggio. Quindi è meglio inviare con un file PROPERTIES (impostazione predefinita) fino a quando si è soddisfatti della progettazione dell'interfaccia utente.

E' possibile utilizzare gli strumenti per creare le strutture HTML per ogni pannello nel file PDML. Al tempo di esecuzione, l'argomento di aiuto corretto viene visualizzato quando l'utente fa clic sul pulsante Aiuto del pannello o preme F1 mentre viene evidenziato uno dei controlli del pannello. E' necessario inserire il contenuto dell'aiuto in punti appropriati nell'HTML, entro le tag <!-- `HELPDOC:SEGMENTBEGIN` --> e <!-- `HELPDOC:SEGMENTEND` -->. Per informazioni più specifiche consultare Modifica dei documenti di aiuto creati dal GUI Builder.

E' possibile creare strutture del codice sorgente per i JavaBean che forniranno i dati per un pannello. Utilizzare la finestra Proprietà del GUI Builder per compilare le proprietà DATACLASS e ATTRIBUTE relative ai controlli che conterranno i dati. La proprietà DATACLASS identifica il nome classe del bean e la proprietà ATTRIBUTE specifica il nome dei metodi gettor/settor implementati dalla classe bean. Una volta aggiunte queste informazioni al file PDML, è possibile utilizzare il GUI Builder per creare strutture del codice sorgente Java e compilarle. In fase di esecuzione, i metodi gettor/settor verranno richiamati per inserire i dati relativi al pannello.

Nota: il numero e il tipo dei metodi gettor/settor dipendono dal tipo di controllo UI a cui sono associati i metodi. I protocolli di metodo per ogni controllo sono documentati nella descrizione di classe per la classe DataBean.

Infine, è possibile serializzare il contenuto del file PDML. La serializzazione produce una rappresentazione binaria compatta di tutte le risorse UI nel file. Ciò migliora notevolmente le prestazioni dell'interfaccia utente, poiché il file PDML non deve essere interpretato per visualizzare i pannelli.

Per riassumere: se è stato creato un file PDML denominato **MyPanels.pdml**, verranno prodotti anche i seguenti file, in base alle opzioni selezionate sugli strumenti:

- **MyPanels.properties** se il bundle di risorse è stato definito come un file PROPERTIES
- **MyPanels.java** e **MyPanels.class** se il bundle di risorse è stato definito come sottoclasse ListResourceBundle
- **<panel name>.html** per ogni pannello nel file PDML, se si è scelto di creare strutture di aiuto in linea
- **<dataclass name>.java** e **<dataclass name>.class** per ogni classe bean univoca specificata nelle proprietà DATACLASS, se si è scelto di creare le strutture codice sorgente per i JavaBean
- **<resource name>.pdml.ser** per ogni risorsa UI definita nel file PDML, se si è scelto di serializzare il contenuto.

Nota: le funzioni relative alla condizione (SELECTED/DESELECTED) non funzioneranno se il nome del pannello è uguale a quello a cui viene associata la funzione relativa alla condizione. Ad esempio, se PANEL1 nel FILE1 ha un riferimento alla condizione associato ad un campo che fa riferimento ad un campo in PANEL1 nel FILE2, l'evento relativo alla condizione non funzionerà. Per risolvere questo problema, rinominare semplicemente PANEL1 in FILE2 e quindi aggiornare l'evento relativo alla condizione nel FILE1 in modo che visualizzi questa modifica.

Esecuzione del Resource Script Converter

Per avviare Resource script converter, richiamare l'interprete Java nel modo seguente:

```
java com.ibm.as400.ui.tools.PDMLViewer
```

Se non è stata impostata la variabile d'ambiente CLASSPATH per contenere i file JAR di Graphical Toolbox, sarà necessario specificarli sulla riga comandi utilizzando l'opzione `classpath`. Consultare Installazione del Graphical Toolbox.

E' possibile anche eseguire Resource Script Converter in modalità batch utilizzando il seguente comando:

```
java com.ibm.as400.ui.tools.RC2XML file [options]
```

Dove *file* è il nome dello script delle risorse (file RC) da elaborare.**Opzioni**

-x name
Il nome del file PDML creato. Come valore predefinito ha il nome del file RC da elaborare.

-p name
Il nome del file PROPERTIES creato. Come valore predefinito ha il nome del file PDML.

-r name
Il nome della sottoclasse ListResourceBundle creata. Come valore predefinito ha il nome del file PDML.

-package name
Il nome del pacchetto a cui verranno assegnate le risorse create. Se non specificato, non verrà creata alcuna istruzione di pacchetto.

-l locale
La locale in cui produrre le risorse create. Se viene specificata una locale, i codici lingua ISO a 2 caratteri e quelli di Paese o regione vengono anteposti al nome del bundle risorse generato.

-h Creare strutture HTML per l'aiuto in linea.

-d Creare strutture del codice sorgente per JavaBeans.

-s Serializzare tutte le risorse.

Come mettere in corrispondenza delle risorse di Windows con PDML

Tutte le finestre di dialogo, i menu e le tabelle di stringa trovate nel file RC verranno convertite nelle corrispondenti risorse Graphical Toolbox nel file PDML creato. E' possibile anche definire le proprietà DATACLASS e ATTRIBUTE per i controlli di Windows che verranno distribuiti al nuovo file PDML utilizzando una semplice convenzione di denominazione quando si creano gli identificativi per le risorse di Windows. Queste proprietà verranno utilizzate per creare strutture del codice sorgente per i JavaBean quando si esegue la conversione.

La convenzione per la denominazione per gli identificativi delle risorse di Windows è:

```
IDCB_<class name>_<attribute>
```

dove `<class name>` è il nome completo della classe bean che si desidera designare come proprietà DATACLASS del controllo e `<attribute>` è il nome della proprietà bean che si desidera designare come proprietà ATTRIBUTE del controllo.

Ad esempio, un campo di testo di Windows con l'ID risorsa `IDCB_com_MyCompany_MyPackage_MyBean_SampleAttribute` produce una proprietà DATACLASS di **com.MyCompany.MyPackage.MyBean** e una proprietà ATTRIBUTE di **SampleAttribute**. Se si decide di creare dei JavaBean quando si esegue la conversione, viene creato il file sorgente Java **MyBean.java**, contenente l'istruzione di pacchetto **package com.MyCompany.MyPackage** ed i metodi gettor e settor per la proprietà **SampleAttribute**.

Visualizzazione dei pannelli al tempo di esecuzione

Il Graphical Toolbox fornisce una API ridistribuibile che i programmi Java possono utilizzare per visualizzare i pannelli di interfaccia utente definiti utilizzando PDML. L'API visualizza i pannelli interpretando il PDML e restituendo l'interfaccia utente tramite le classi Foundation Java.

L'ambiente del tempo di esecuzione di Graphical Toolbox fornisce i seguenti servizi:

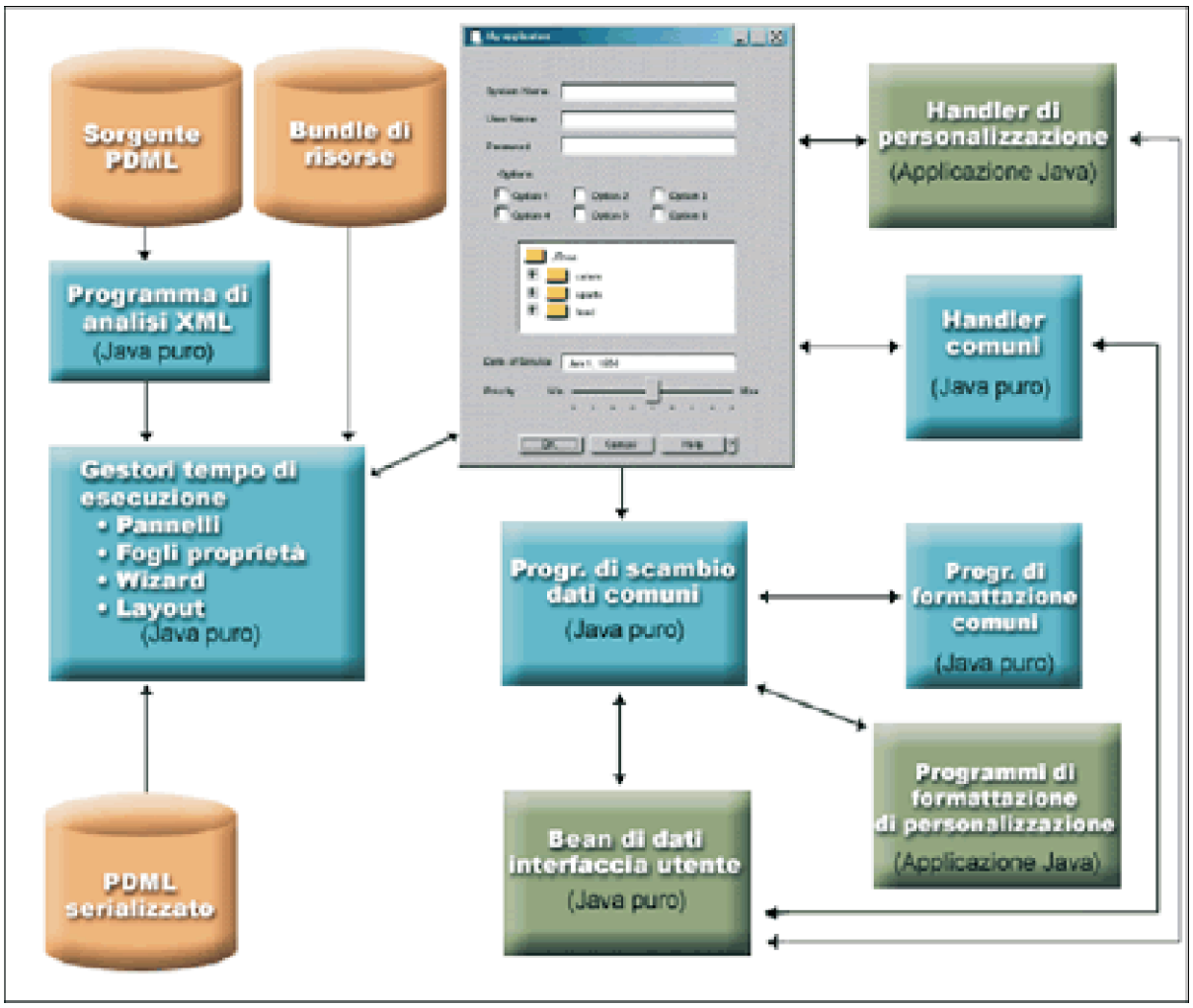
- Gestisce tutti gli scambi di dati tra i controlli dell'interfaccia utente ed i JavaBean identificati nel PDML.
- Esegue la convalida dei dati utente per tipi di dati character e integer comuni e definisce un'interfaccia che consente di realizzare una convalida personalizzata. Se i dati vengono ritenuti non validi, viene inviato all'utente un messaggio d'errore.
- Definisce l'elaborazione standardizzata per eventi di commit, di annullamento e di aiuto e fornisce una framework per la gestione di eventi personalizzati.

- Gestisce le interazioni tra i controlli d'interfaccia utente in base alle informazioni sullo stato definite nel PDML. (Ad esempio, è possibile che si desideri disabilitare un gruppo di controlli ogni volta che l'utente seleziona un particolare pallino.)

Il pacchetto com.ibm.as400.ui.framework.java contiene l'API del tempo di esecuzione di Graphical Toolbox.

Gli elementi dell'ambiente del tempo di esecuzione di Graphical Toolbox vengono mostrati nella Figura 1. Il programma Java è un client di uno o più degli oggetti nella casella **Gestori tempo di esecuzione**.

Figura 1: ambiente tempo di esecuzione di Graphical Toolbox



Esempi

Supponiamo che il pannello **MyPanel** sia definito nel file **TestPanels.pdml** e che un file delle proprietà **TestPanels.properties** sia associato alla definizione del pannello.Entrambi i file risiedono nell'indirizzario **com/ourCompany/ourPackage**, a cui si può accedere da un'indirizzario definito nel percorso classe o da un file ZIP o JAR definito nel percorso classe.

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per informazioni legali importanti.

Esempio: creazione e visualizzazione di un pannello

Il codice che segue crea e visualizza il pannello:

```
import com.ibm.as400.ui.framework.java.*;

// Creare il gestore pannelli. Parametri:
// 1. Nome risorsa della definizione di pannello
// 2. Nome del pannello
// 3. Elenco dei DataBean omessi

PanelManager pm = null;
try {
  pm = new PanelManager("com.ourCompany.ourPackage.TestPanels", "MyPanel", null);
}

catch (DisplayManagerException e) {
  e.displayUserMessage(null);
  System.exit(-1);
}

// Visualizzare il pannello
pm.setVisible(true);
```

Esempio: creazione di una finestra di dialogo

Dopo aver implementato i DataBean che forniscono i dati al pannello ed aver identificato gli attributi nel PDML, si può utilizzare il codice che segue per creare una finestra di dialogo completamente funzionante:

```
import com.ibm.as400.ui.framework.java.*;
import java.awt.Frame;

// Avviare gli oggetti che forniscono dati al pannello
TestDataBean1 db1 = new TestDataBean1();
TestDataBean2 db2 = new TestDataBean2();

// Inizializzare gli oggetti
db1.load();
db2.load();

// Impostare l'inoltro degli oggetti alla framework UI
DataBean[] dataBeans = { db1, db2 };

// Creare il gestore pannelli. Parametri:// 1. Nome risorsa della definizione di pannello
// 2. Nome del pannello
// 3. Elenco dei DataBean
// 4. Finestra frame proprietario

Frame owner;
...
PanelManager pm = null;
try {
  pm = new PanelManager("com.ourCompany.ourPackage.TestPanels", "MyPanel", dataBeans, owner);
}

catch (DisplayManagerException e) {
  e.displayUserMessage(null);
  System.exit(-1);
}

// Visualizzare il pannello
pm.setVisible(true);
```

Esempio: utilizzo del gestore pannelli dinamico

Al gestore pannelli esistente è stato aggiunto un nuovo servizio. Il gestore pannelli dinamico dimensiona dinamicamente il pannello durante il tempo di esecuzione. Si consideri ancora una volta l'esempio in **MyPanel**, utilizzando il gestore pannelli dinamico:

```
import com.ibm.as400.ui.framework.java.*;

// Creare il gestore pannelli dinamico. Parametri:
// 1. Nome risorsa della definizione di pannello
// 2. Nome del pannello
// 3. Elenco dei DataBean omessi

DynamicPanelManager dpm = null;
try {
  pm = new DynamicPanelManager("com.ourCompany.ourPackage.TestPanels", "MyPanel", null);
}

catch (DisplayManagerException e) {
  e.displayUserMessage(null);
  System.exit(-1);
}

// Visualizzare il pannello
pm.setVisible(true);
```

Quando si crea l'istanza di questa applicazione del pannello, è possibile vedere la funzione di dimensionamento dinamico del pannello. Spostare il cursore sul bordo del pannello della GUI e, quando sono visualizzate le frecce di dimensionamento, è possibile modificare la dimensione del pannello.

Descrizione lunga della Figura 1: ambiente tempo di esecuzione del Graphical Toolbox (rzahh504.gif)

In IBM Toolbox per Java: visualizzare i pannelli al tempo di esecuzione

Questa figura illustra come gli elementi dell'ambiente tempo di esecuzione del Graphical Toolbox interagiscono con il codice di applicazione.

Descrizione

La figura è composta da diverse caselle di forme, dimensioni e e colori differenti collegate le une con le altre da linee che terminano con delle frecce su uno o entrambe gli estremi.

Per visualizzare la figura, è utile dividerla in tre colonne e quattro righe, numerando le aree in sequenza da sinistra in alto a destra in basso. Ad esempio, la prima riga contiene le aree 1, 2 e 3; la seconda le aree 4, 5 e 6; e così via.

- L'immagine di una finestra di dialogo che occupa le aree 2 e 5 rappresenta l'interfaccia GUI per il programma Java. La casella di dialogo dispone di una varietà di opzioni, quali caselle di spunta, campi di testo e così via.
- Due cilindri rossicci nella parte superiore dell'area 1 sono etichettati Sorgente PDML e Bundle di risorse. Questi cilindri rappresentano il sorgente PDML e i file risorse Java che risiedono nella memoria intermedia.
- Un cilindro rossiccio nell'area 10 etichettato PDML serializzato rappresenta uno o più file PDML serializzati che risiedono in un supporto magnetico di memoria.
- I cinque rettangoli blu che circondano la porzione inferiore della casella di dialogo rappresentano i componenti del Graphical Toolbox. Iniziando dal rettangolo all'estrema sinistra e muovendosi in senso antiorario, essi vengono etichettati:
  - Programma di analisi XML (Java puro) nell'area 4, che rappresenta l'IBM XML Parser.
  - Gestori tempo di esecuzione (Java puro) nell'area 7. Il programma Java è un client di uno o più degli oggetti contenuti nei Gestori tempo di esecuzione: pannelli, fogli delle proprietà, wizard e layout.

  - Programma di scambio dati comune (Java puro) nell'area 8.
  - Programmi di formattazione comuni (Java puro) nell'area 9.
  - Handler comuni (Java puro) nell'area 6.
- I tre rettangoli verdi rappresentano il codice fornito dal programmatore dell'applicazione e sono etichettati:
  - Handler di personalizzazione (Applicazione Java) nell'area 3
  - Programmi di formattazione di personalizzazione (Applicazione Java) nell'area 12
  - Bean di dati dell'interfaccia utente (Java puro) nell'area 11
- Le linee collegano molti dei modelli:
  - Una linea che ha un freccia singola (solo su una estremità) indica un'azione. Le linee a freccia singola puntano verso una funzione o un componente che utilizza l'oggetto da cui origina la linea. Nella seguente descrizione, la parola "utilizzare" significa che una linea a freccia singola punta verso un oggetto partendo dal componente che agisce su di esso.
  - Una linea che ha due frecce (una per ogni estremità) indica un'interazione. Queste linee collegano gli oggetti che condividono uno scambio a doppio senso di informazioni. Nella seguente descrizione, la parola "interagire" significa che i componenti sono collegati da una linea a due frecce.

L'interfaccia GUI per il programma Java (l'immagine della casella nelle aree 2 e 5) interagisce con i Gestori tempo di esecuzione per il Graphical Toolbox (il rettangolo blu nell'area 7).

I Gestori tempo di esecuzione, che sono Java puro, contengono pannelli, fogli delle proprietà, wizard e il layout della GUI. Per creare la GUI, i Gestori tempo di esecuzione utilizzano un bundle di risorse Java (uno dei due cilindri rossicci nell'area 1) e dati PDML. I Gestori tempo di esecuzione possono elaborare dati PDML in due modi:

- Utilizzando file PDML serializzati (il cilindro nell'area 10)
- Utilizzando IBM iSeries XML Parser (il rettangolo blu nell'area 4), che a sua volta utilizza (analizza) i file sorgente PDML (uno dei due cilindri rossicci nell'area 1)

Il programma Java abilitato per la GUI opera sui dati in base ad una delle seguenti modalità:

- Con l'interfaccia GUI che interagisce con gli handler di personalizzazione (il rettangolo verde nell'area 3) e con gli handler comuni (il rettangolo blu nell'area 6)
- Con il programma di scambio dati comune (il rettangolo blu nell'area 8) che utilizza l'interfaccia GUI per richiamare informazioni.

Gli Handler personalizzazione, gli Handler comuni e il programma di scambio dati comuni interagiscono tutti con i bean di dati dell'interfaccia utente (il rettangolo verde nell'area 11), passando informazioni in entrambi i sensi. Il programma di scambio di dati comune interagisce con i programmi di formattazione comuni (il rettangolo blu dell'area 9) e con in programmi di formattazione di personalizzazione (il rettangolo verde nell'area 12) per convertire dati in formati appropriati per il bean di dati dell'interfaccia utente.

Modifica dei documenti di aiuto creati dal GUI Builder

Per ognuno dei file progetto PDML, GUI Builder crea una struttura di aiuto e la immette in un documento HTML singolo. Prima dell'utilizzo, questo file HTML viene diviso in file HTML ad argomento singolo per ogni finestra di dialogo del progetto PDML. Ciò fornisce all'utente un aiuto approssimativo per ogni argomento e consente di gestire solo alcuni grandi file di aiuto.

Il documento di aiuto è un file HTML valido e può essere visualizzato in qualsiasi browser e modificato utilizzando la maggior parte degli editor HTML. Le tag che definiscono le sezioni in un documento di aiuto vengono incorporate all'interno dei commenti, in modo da non essere visualizzate nel browser. Le tag di commento vengono utilizzate per dividere il documento di aiuto in varie sezioni:

- Intestazione
- Sezione di argomento per ogni finestra di dialogo
- Sezione di argomento per ogni controllo con l'aiuto abilitato
- Piè di pagina

Inoltre, è possibile aggiungere delle sezioni di argomento supplementari prima del piè di pagina per fornire ulteriori informazioni o informazioni comuni. Quando si creano l'intestazione e il piè di pagina, le sezioni di aiuto dispongono solo del corpo html fino a quando vengono divise. Quando il documento di aiuto viene diviso il processore aggiunge un'intestazione e un piè di pagina alla sezione di argomento per rendere un file HTML completo. L'intestazione e il piè di pagina dal documento di aiuto vengono utilizzati come intestazione e piè di pagina predefiniti. Tuttavia, è possibile sostituire l'intestazione predefinita con una personalizzata.

All'interno del documento di aiuto

Le sezioni che seguono spiegano le parti del documento di aiuto:

Intestazione

La fine della sezione dell'intestazione viene visualizzata dalla tag che segue:

```
<!-- HELPDOC:HEADEREND -->
```

Se si desidera sostituire l'intestazione predefinita per tutti gli argomenti individuali quando vengono divisi, utilizzare la parola chiave HEADER e fornire il nome di un frammento html da includere.Ad esempio:

```
<!-- HELPDOC:HEADEREND HEADER="defaultheader.html" -->
```

Segmento di argomento

Ogni argomento delimitato dalle tag che seguono:

```
<!-- HELPDOC:SEGMENTBEGIN -->
```

and

```
<!-- HELPDOC:SEGMENTEND -->
```

Subito dopo la tag SEGMENTBEGIN vi è una tag ancora che denomina il segmento. Fornisce inoltre il nome file del documento HTML creato quando il documento di aiuto viene diviso. Il nome del segmento unisce l'identificativo del pannello, l'identificativo del controllo e l'estensione file futura. (html). Ad esempio: i segmenti "MY_PANEL.MY_CONTROL.html" per i pannelli dispongono solo dell'identificativo del pannello e dell'estensione file futura.

Il generatore di aiuto posizionerà il testo nel documento indicando dove è possibile immettere le informazioni di aiuto:

```
<!-- HELPDOC:SEGMENTBEGIN PDMLSYNCH="YES" --><A NAME="MY_PANEL.MY_CONTROL.html"></A>
<H2>My favorite control</H2>
Insert help for "My favorite control" here.
<P><!-- HELPDOC:SEGMENTEND -->
```

E' possibile aggiungere tag HTML 2.0 supplementari, se necessario, dopo la tag ancora e prima della tag SEGMENTEND.

La tag PDMLSYNCH ha la funzione di controllare quanto vicino si trova il segmento ai controlli definiti in PDML. Se PDMLSYCH corrisponde a "YES", il segmento del Documento di aiuto verrà rimosso se il controllo con lo stesso nome viene rimosso in PDML. PDMLSYNCH="NO" indica che l'argomento deve essere conservato nel Documento di aiuto a prescindere dall'esistenza o meno di un controllo corrispondente nel PDML. Ad esempio, ciò viene utilizzato quando si creano ulteriori argomenti, comuni o più specifici.

L'aiuto creato per un pannello, dispone di collegamenti a ciascun controllo abilitato per l'aiuto nel pannello. Tali collegamenti vengono creati con un riferimento ancora locale, in modo tale da poterli verificare come collegamenti interni in un browser standard. Quando il Documento di aiuto viene suddiviso, il processore rimuove il simbolo "#" da tali collegamenti interni rendendoli così esterni nei file HTML del singolo argomento risultanti. Poiché è possibile che si desideri disporre di collegamenti interni all'interno di un argomento, il processore rimuove solo qualsiasi simbolo "#" che lo precede quando il riferimento dispone di ".html".

Se si desidera sostituire l'intestazione predefinita per un particolare argomento, utilizzare la parola chiave HEADER e fornire il nome di un frammento html da includere. Ad esempio:

```
<!-- HELPDOC:SEGMENTBEGIN PDMLSYNCH="YES" HEADER="specialheader.html" -->
```

Piè di pagina

Il piè di pagina nel documento di aiuto inizia con la tag che segue:

```
<!-- HELPDOC:FOOTERBEGIN -->
```

Il piè di pagina standard è </BODY></HTML>Tale piè di pagina viene aggiunto a ogni file HTML.

Aggiunta dei collegamenti

E' possibile aggiungere collegamenti ad ogni URL esterna o interna così come qualsiasi altro segmento. Tuttavia, è necessario seguire alcune convenzioni:

- Le URL esterne vengono utilizzate in modo standard. Ciò include collegamenti interni ad URL esterne
- I collegamenti interni nello stesso argomento vengono scritti nella forma standar ma non devono avere ".html" come parte del nome della tag. Ciò perché il processore del documento di aiuto presuppone che ogni collegamento con .html dovrà essere un collegamento esterno quando gli argomenti vengono divisi. Perciò, rimuove il simbolo "#" che lo precede.
- I collegamenti ad altri segmenti dell'argomento devono essere scritti con un simbolo "#" che li precede, sebbene siano un riferimento ancora interno.
- E' inoltre possibile creare collegamenti interni con altri segmenti di argomento. Durante il processo viene rimosso soltanto il simbolo "#" iniziale.

Nota:

- Al momento dell'esecuzione, la classe PanelManager cerca i file di aiuto in un sottoindirizzario con lo stesso nome del file PDML. Quando il processore divide il documento di aiuto, crea questo sottoindirizzario per impostazione predefinita e posiziona in esso i file HTML risultanti.
- Il processore non provvede ad alcun adattamento per i riferimenti URL che rappresentano collegamenti relativi. Quando ci si collega da un file di argomento singolo, ogni collegamento relativo effettuerà la ricerca dal nuovo sottoindirizzario. Perciò, l'utente avrà bisogno di collocare copie di risorse, ad esempio immagini, dove possano essere trovate o utilizzare "../" nel percorso per effettuare la ricerca dall'indirizzario del pannello.

Esecuzione di una modifica utilizzando un editor visuale

E' possibile modificare il contenuto di aiuto in quasi tutti gli editor HTML visuali. Poiché le tag HELPDOC sono commenti, potrebbero non essere evidenti in alcuni editor. Per comodità, viene aggiunta una regola orizzontale alla struttura dell'aiuto prima della tag SEGMENTBEGIN e dopo la tag SEGMENTEND. Tali regole orizzontali, forniscono un'indicazione della visuale più chiara dell'intero segmento in un editor visuale. Se viene selezionato un segmento con lo scopo di spostarlo, copiarlo o cancellarlo, selezionare le regole orizzontali circostanti per assicurarsi di aver incluso le tag SEGMENTBEGIN e SEGMENTEND nella selezione. Tali regole orizzontali non vengono copiate nei file HTML individuali finali.

Creazione di argomenti aggiuntivi

E' possibile creare segmenti dell'argomento aggiuntivi nel Documento di aiuto. Tuttavia, si consiglia di copiare un altro segmento. Quando si copia il segmento, è necessario copiare le regole orizzontali prima della tag SEGMENTBEGIN e dopo la tag SEGMENTEND. Questa operazione renderà più semplice una modifica visuale futura e aiuterà ad evitare tag non corrispondenti. Per migliori risultati, utilizzare i suggerimenti che seguono:

- Il nome dell'ancora deve corrispondere al nome che si desidera assegnare al file singolo risultante quando il Documento di aiuto viene suddiviso. Deve terminare con ".html".
- Utilizzare la parola chiave PDMLSYNCH="NO" nella tag SEGMENTBEGIN per evitare che il segmento venga rimosso se la struttura di aiuto viene creata nuovamente.
- Ogni riferimento al nuovo argomento sarà effettuato come un collegamento interno nel documento di aiuto, preceduto da un simbolo "#". Tale simbolo "#" verrà rimosso nelle elaborazioni successive quando i segmenti vengono suddivisi in file singoli.

Controllo dei collegamenti

Per la maggior parte dei documenti scritti, è possibile controllare i collegamenti visualizzando il documento in un browser web e selezionando diversi collegamenti. Nel documento di aiuto singolo, i collegamenti si trovano ancora nello relativo modulo interno.

A completamento avvenuto o quando si desidera verificare l'applicazione per cui si sta sviluppando l'aiuto, è necessario dividere il documento di aiuto in file singoli. E' possibile effettuare ciò con il Documento di aiuto per l'elaborazione HTML.

Se è necessario ricreare il documento di aiuto dopo la modifica, ciò che è stato scritto sarà conservato. E' possibile che si desideri ricreare il documento di aiuto se si aggiungono nuovi controlli dopo la creazione della struttura di origine dell'aiuto. In questo caso, il generatore di aiuto controlla se esiste già un documento di aiuto prima di creare una nuova struttura. Se ne trova una, conserva qualsiasi segmento esistente e aggiunge i nuovi controlli.

Utilizzo del Graphical Toolbox in un browser

E' possibile utilizzare il Graphical Toolbox per creare i pannelli per le applet Java che si eseguono in un browser web. Questa sezione descrive il modo di convertire il pannello semplice dall'Esempio di Graphical Toolbox per l'esecuzione in un browser. I livelli minimi del browser supportati sono Netscape 4.05 e Internet Explorer 4.0. Per evitare di riscontrare idiosincrasie di singoli browser, si consiglia di eseguire le applet utilizzando il Modulo aggiuntivo Java di Sun. In caso contrario, sarà necessario creare file JAR siglati per Netscape e file CAB siglati separati per Internet Explorer.

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per informazioni legali importanti.

Creazione dell'applet

Il codice per la visualizzazione di un pannello in un'applet è quasi identico al codice utilizzato nell'esempio dell'applicazione Java, ma come prima cosa, tale codice deve essere nuovamente compresso nel metodo **init** di una sottoclasse **JApplet**. Inoltre, è necessario aggiungere un codice per assicurarsi che il pannello dell'applet sia delle dimensioni specificate nella definizione PDML del pannello. Di seguito viene riportato il codice sorgente dell'applet di esempio, **SampleApplet.java**.

```
import com.ibm.as400.ui.framework.java.*;

import javax.swing.*;
import java.awt.*;
import java.applet.*;
import java.util.*;

public class SampleApplet extends JApplet
{
        // Quanto di seguito riportato è necessario per mantenere la dimensione del pannello
        private PanelManager        m_pm;
        private Dimension           m_panelSize;

        // Definire un'eccezione da avviare le caso in cui si verifichi un errore
        class SampleAppletException extends RuntimeException {}

        public void init()
    {
        System.out.println("In init!");

                // Tenere traccia dei parametri dell'applet
                System.out.println("SampleApplet code base=" + getCodeBase());
                System.out.println("SampleApplet document base=" + getDocumentBase());

                // Controllare per assicurarsi di essere in esecuzione su una JVM compatibile con Swing 1.1
                if (System.getProperty("java.version").compareTo("1.1.5") < 0)
```

```
                        throw new IllegalStateException("SampleApplet cannot run on Java VM version " +
                                        System.getProperty("java.version") +
                                        " - requires 1.1.5 or higher");

                // Avviare l'oggetto bean che fornisce i dati al pannello
                SampleBean bean = new SampleBean();

                // Inizializzare l'oggetto
                bean.load();

                // Impostare l'inoltro del bean al gestore pannelli
                DataBean[] beans = { bean };

                // Aggiornare la barra di stato
        showStatus("Loading the panel definition...");

        // Creare il gestore pannelli.
Parametri:
                // 1. File PDML come nome risorsa
                // 2. Nome del pannello da visualizzare
                // 3. Elenco degli oggetti di dati che forniscono i dati al pannello
                // 4. Il contenuto del pannello dell'applet

                try { m_pm = new PanelManager("MyGUI", "PANEL_1", beans, getContentPane()); }
                catch (DisplayManagerException e)
        {
                        // Si è verificato un errore, quindi visualizzare un messaggio e uscire
            e.displayUserMessage(null);
                        throw new SampleAppletException();
        }

                // Identificare l'indirizzario in cui risiede l'aiuto in linea
                m_pm.setHelpPath("http://MyDomain/MyDirectory/");

        // Visualizzare il pannello
                m_pm.setVisible(true);
    }

        public void start()
    {
        System.out.println("In start!");

                // Riportare il pannello alla sua dimensione predefinita
                m_panelSize = m_pm.getPreferredSize();
                if (m_panelSize != null)
        {
                        System.out.println("Resizing to " + m_panelSize);
                        resize(m_panelSize);
        }
                                else
                        System.err.println("Error: getPreferredSize returned null");
    }

        public void stop()
    {
        System.out.println("In stop!");
    }

        public void destroy()
    {
        System.out.println("In destroy!");
    }

        public void paint(Graphics g)
    {
                // Chiamare prima il parent
                super.paint(g);
```

```
                // Conservare la dimensione predefinita del pannello su un repaint
                if (m_panelSize != null)
                        resize(m_panelSize);
    }
}
```

Il pannello del contenuto dell'applet viene passato al Graphical Toolbox come un contenitore di cui configurare il layout. Nel metodo **start**, il pannello applet è impostato con la corretta dimensione, quindi viene sovrapposto il metodo **paint** per conservare la dimensione del pannello quando la finestra del browser viene ridimensionata.

Quando si esegue il Graphical Toolbox in un browser, non è possibile accedere ai file HTML per l'aiuto in linea del pannello da un file JAR. Questi si devono trovare come file separati nell'indirizzario in cui si trova l'applet. La chiamata a **PanelManager.setHelpPath** identifica quest'indirizzario per il Graphical Toolbox, in modo da poter individuare i file d'aiuto.

Tag HTML

Poiché si consiglia l'utilizzo del Modulo aggiuntivo Java di Sun per consentire il giusto livello dell'ambiente tempo di esecuzione di Java, l'HTML per l'identificazione dell'applet del Graphical Toolbox non è semplice come lo si desidera. Fortunatamente, si può riutilizzare la stessa maschera HTML, solo con lievi modifiche, per altri applet. La markup è stata progettata per essere interpreta sia in Netscape Navigator che in Internet Explorer e ciò da luogo ad una richiesta di scaricare il Modulo aggiuntivo di Java dal sito web di Sun qualora non sia già stato installato sul sistema dell'utente. Per informazioni dettagliate sul funzionamento del Modulo aggiuntivo di Java, consultare Specifiche HTML del Modulo aggiuntivo di Java. 

Questo è il codice HTML per l'applet di esempio, nel file **MyGUI.html**:

```
<html>

<head>
<title>Graphical Toolbox Demo</title>
</head>

<body>
<h1>Graphical Toolbox Demo Using Java(TM) Plug-in</h1>
<p>

<!-- INIZIO TAG APPLET MODULO AGGIUNTIVO JAVA(TM) -->

<!-- Le seguenti tag usano una sintassi speciale che consente a Netscape e Internet Explorer di caricare -->
<!-- il Modulo aggiuntivo Java ed eseguire l'applet nel JRE di tale modulo. Non cambiare la sintassi.            -->
<!-- Per informazioni andare all'indirizzo http://java.sun.com/products/jfc/tsc/swingdoc-current/java_plug_in.html.-->

<OBJECT classid="clsid:8AD9C840-044E-11D1-B3E9-00805F499D93"
        width="400"
        height="200"
        align="left"
        codebase="http://java.sun.com/products/plugin/1.1.3/jinstall-113-win32.cab#Version=1,1,3,0">
    <PARAM name="code"     value="SampleApplet">
    <PARAM name="codebase" value="http://www.mycompany.com/~auser/applets/">
    <PARAM name="archive"  value="MyGUI.jar,jui400.jar,util400.jar,x4j400.jar">
    <PARAM name="type"     value="application/x-java-applet;version=1.1">

    <COMMENT>
    <EMBED type="application/x-java-applet;version=1.1"
           width="400"
           height=200"
           align="left"
           code="SampleApplet"
```

```
          codebase="http://www.mycompany.com/~auser/applets/"
          archive="MyGUI.jar,jui400.jar,util400.jar,x4j400.jar"
          pluginspage="http://java.sun.com/products/plugin/1.1.3/plugin-install.html">
        <NOEMBED>
    </COMMENT>
                No support for JDK 1.1 applets found!
        </NOEMBED>
    </EMBED>
</OBJECT>

<!-- FINE TAG APPLET MODULO AGGIUNTIVO JAVA(TM) -->

<p>
</body>
</html>
```

E' importante che le informazioni sulla versione vengano impostate per 1.1.3.

Nota: in questo esempio, il file JAR del programma di analisi XML, **x4j400.jar**, è memorizzato nel server Web. E' possibile utilizzare altri programmi di analisi XML. Per ulteriori informazioni, consultare "Programma di analisi XML e processore XSLT" a pagina 417. Questo è necessario solo quando si inserisce il file PDML come parte dell'installazione dell'applet. Per motivi di prestazioni, normalmente si dovrebbero *serializzare* le definizioni del pannello in modo che il Graphical Toolbox non debba interpretare il PDML durante il tempo di esecuzione.Questo migliora molto le prestazioni dell'interfaccia utente creando rappresentazioni binarie compatte dei pannelli. Per ulteriori informazioni consultare la descrizione dei file creati dagli strumenti.

Installazione ed esecuzione dell'applet

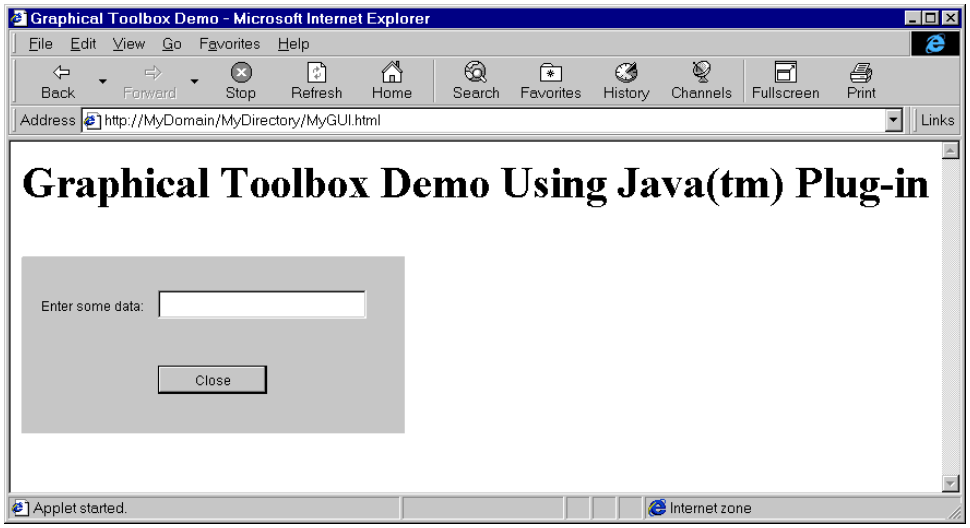
Installare l'applet sul server web preferito attenendosi alle seguenti istruzioni:

1. Compilare **SampleApplet.java**.
2. Creare un file JAR denominato **MyGUI.jar** per contenere i binari dell'applet.Questi includono i file di classe creati durante la compilazione di **SampleApplet.java** e **SampleBean.java**, del file PDML **MyGUI.pdml** e del bundle di risorse **MyGUI.properties**.
3. Copiare il nuovo file JAR su un indirizzario a propria scelta nel proprio server web. Copiare i file HTML che contengono l'aiuto in linea nell'indirizzario del server.
4. Copiare i file JAR del Graphical Toolbox nell'indirizzario del server.
5. Infine, copiare il file HTML **MyGUI.html** che contiene l'applet incorporata nell'indirizzario del server.

Suggerimento: Durante la verifica delle applet, assicurarsi di aver eliminato i file JAR del Graphical Toolbox dalla variabile di ambiente CLASSPATH nella propria stazione di lavoro. Altrimenti, saranno visualizzati messaggi d'errore in cui viene riportato che non si possono individuare le risorse dell'applet sul server.

Adesso è possibile eseguire l'applet. Puntare il browser web su **MyGUI.html** nel server.Se il Modulo aggiuntivo di Java non è ancora stato installato, verrà richiesto se si desidera installarlo. Una volta installato il Modulo aggiuntivo ed avviata l'applet, il pannello del browser apparirà come nella Figura 1:

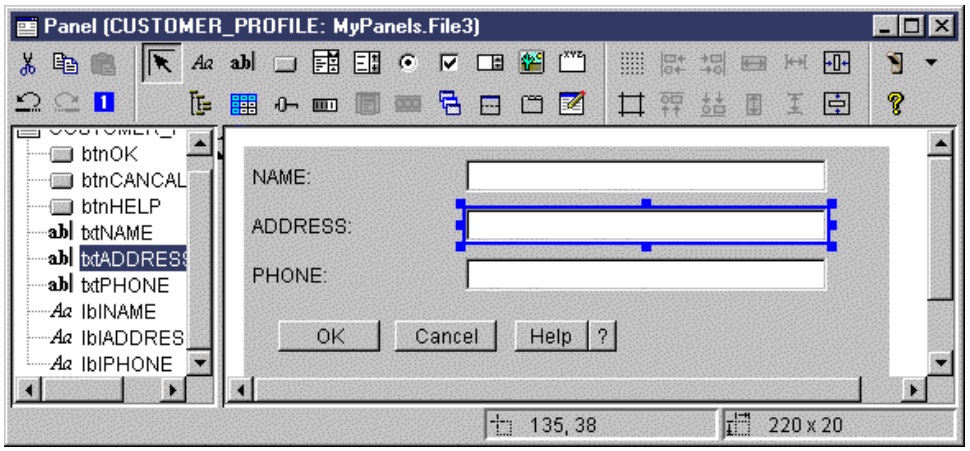
Figura 1: esecuzione dell'applet di esempio in un browser



Barra degli strumenti Programma di creazione pannello GUI Builder

La figura 1 visualizza la finestra Programma di creazione pannello GUI Builder. La figura 1 che segue è un elenco che mostra ogni icona dello strumento Programma di creazione pannello e ne descrive la funzione.

Figura 1: finestra Programma di creazione pannello GUI Builder



 Fare clic sul Puntatore per spostare o ridimensionare un componente su un pannello.

 Fare clic su Etichetta per inserire un'etichetta statica su un pannello.

 Fare clic su Testo per inserire una casella di testo su un pannello.

 Fare clic su Pulsante per inserire un pulsante su un pannello.

 Fare clic su Casella combinata per inserire una casella di elenco a discesa su un pannello.

 Fare clic su Casella di elenco per inserire una casella di elenco su un pannello.

 Fare clic su Pallino per inserire un pallino su un pannello.

 Fare clic su Casella di spunta per inserire una casella di spunta su un pannello.

 Fare clic su Spinner per inserire uno spinner su un pannello.

 Fare clic su Immagine per inserire un'immagine su un pannello.

 Fare clic su Barra dei menu per inserire una barra dei menu su un pannello.

 Fare clic su Casella di gruppo per inserire una casella di gruppo etichettata su un pannello.

 Fare clic su Albero per inserire un albero gerarchico su un pannello.

 Fare clic su Tabella per inserire una tabella su un pannello.

 Fare clic su Dispositivo di scorrimento per inserire un dispositivo di scorrimento regolabile su un pannello.

 Fare clic su Barra di avanzamento per inserire una barra di avanzamento su un pannello.

 Fare clic su Pannello sovrapposto per inserire un pannello sovrapposto su un pannello.Un pannello sovrapposto contiene uno stack di pannelli. L'utente può selezionare ogni pannello ma solo il pannello selezionato è completamente visibile.

 Fare clic su Pannello suddiviso per inserire un Pannello suddiviso su un pannello.Un pannello suddiviso è un pannello diviso in due pannelli orizzontali o verticali.

 Fare clic su Pannello con separatore per inserire un pannello con separatore su un pannello.Un pannello con separatore contiene una raccolta di pannelli con separatori nella parte superiore. L'utente fa clic su un separatore per visualizzare il contenuto di un pannello. Il titolo del pannello viene utilizzato come il testo di un separatore.

 Fare clic su Personalizza per inserire un componente interfaccia utente definito dalla personalizzazione su un pannello.

 Fare clic su Barra degli strumenti per inserire una barra degli strumenti su un pannello.

 Fare clic su Alterna griglia per abilitare una griglia su un pannello.

 Fare clic su Allinea in alto per allineare più componenti su un pannello con il margine superiore di un componente specifico o primario.

 Fare clic su Allinea in basso per allineare più componenti su un pannello con il margine inferiore di un componente specifico o primario.

 Fare clic su Livella altezza per livellare l'altezza di più componenti rispetto all'altezza di un componente specifico o primario.

 Fare clic su Centra verticalmente per centrare verticalmente un componente selezionato rispetto al pannello.

 Fare clic su Alterna margini per visualizzare i margini del pannello.

 Fare clic su Allinea a sinistra per allineare più componenti su un pannello con il bordo sinistro di un componente specifico o primario.

 Fare clic su Allinea a destra per allineare più componenti su un pannello con il bordo sinistro di un componente specifico o primario.

 Fare clic su Livella ampiezza per livellare l'ampiezza di più componenti rispetto all'ampiezza di un componente specifico o primario.

 Fare clic su Centra orizzontalmente per centrare un componente selezionato orizzontalmente nel pannello.

 Fare clic su Taglia per tagliare i componenti del pannello.

 Fare clic sul pulsante Copia per copiare i componenti del pannello.

 Fare clic su Incolla per incollare i componenti del pannello tra diversi pannelli o file.

 Fare clic su Annulla per annullare l'ultima operazione.

 Fare clic su Ripristina per ripristinare l'ultima operazione.

 Fare clic su Ordina separatore per controllare l'ordine di selezione di ogni componente del pannello quando l'utente preme l'icona Separatore per navigare sul pannello.

 Fare clic su Anteprima per visualizzare un'anteprima del pannello.

 Fare clic su Aiuto per ottenere informazioni più specifiche su Graphical Toolbox.

Bean IBM Toolbox per Java

JavaBeans(TM) sono componenti software riutilizzabili scritti in Java. Il componente è una parte di codice del programma che fornisce una unità funzionale e ben definita che può essere piccola quanto un'etichetta o grande quanto un'applicazione.

I JavaBean possono essere componenti visuali o non grafici. I JavaBean non grafici hanno tuttavia una rappresentazione visuale, come un'icona o un nome, per consentire la manipolazione visuale.

Tutte le classi pubbliche IBM Toolbox per Java sono JavaBeans. Queste classi sono state create per gli standard Javasoft JavaBean; esse funzionano come componenti riutilizzabili. Le proprietà e i metodi per un bean IBM Toolbox per Java sono uguali ai metodi della classe.

I JavaBean possono essere utilizzati in un programma dell'applicazione oppure possono essere utilizzati in modo visuale negli strumenti del programma di creazione, ad esempio il prodotto IBM VisualAge per Java.

Esempi

I seguenti esempi mostrano come utilizzare JavaBeans nel programma e come creare un programma da JavaBeans utilizzando un programma di creazione del bean visuale:

"Esempio: codice bean IBM Toolbox per Java" a pagina 539

"Esempio: creazione di bean con un programma di creazione del bean visuale" a pagina 541

JDBC

JDBC(TM) è un'API (application programming interface) inclusa nella piattaforma Java che consente ai programmi Java di collegarsi ad una ampia gamma di database.

L'unità di controllo JDBC di IBM Toolbox per Java consente di utilizzare le interfacce API JDBC per emettere istruzioni SQL (structured query language) ed elaborare i risultati dai database sul server. E' inoltre possibile utilizzare l'unità di controllo JDBC di IBM Developer Kit per Java, denominata unità di controllo JDBC 'nativo':

- Utilizzare l'unità di controllo JDBC di IBM Toolbox quando il programma Java si trova su un sistema e i file del database si trovano su un altro, come in un ambiente client/server
- Utilizzare l'unità di controllo JDBC nativo quando sia il programma Java che i file database si trovano nello stesso server iSeries

Per ulteriori informazioni sugli esempi e sulle classi JDBC di IBM Toolbox per Java, sui miglioramenti, sulle proprietà JDBC e sui tipi SQL non supportati, consultare le seguenti pagine:

"Classi JDBC" a pagina 63

"Aggiornamenti del supporto JDBC di IBM Toolbox per Java" a pagina 325

"Proprietà JDBC di IBM Toolbox per Java" a pagina 328

"Tipi SQL JDBC" a pagina 345

Differenti versioni di JDBC

Esistono differenti versioni API di JDBC e l'unità di controllo JDBC IBM Toolbox per Java supporta le seguenti versioni:

- JDBC 1.2 API (pacchetto java.sql) è incluso nell'API principale 1.1 di Java Platform 1.1 e JDK 1.1.
- API principale di JDBC 2.1 (pacchetto java.sql) è incluso in Java 2 Platform, Standard Edition (J2SE) e in Java 2 Platform Enterprise Edition (J2EE).

- L'API JDBC 2.0 Optional Package (pacchetto javax.sql) è incluso in J2EE ed è disponibile sotto forma di download separato da Sun.Queste estensioni erano precedentemente chiamate JDBC 2.0 Standard Extension API.
- API di JDBC 3.0 (pacchetti java.sql e javax.sql) sono incluse in J2SE, Versione 1.4.

Aggiornamenti del supporto JDBC di IBM Toolbox per Java

Gli aggiornamenti delle funzioni JDBC per OS/400 Versione 5 Release 3 includono:

- supporto UTF-8 e UTF-16
- supporto BINARY e VARBINARY
- supporto IDP (Increased Decimal Precision)
- supporto LOB 2 GB:
- supporto cursore non sensibile
- supporto MQT (Materialized Query Table)

Per informazioni sugli aggiornamenti delle funzioni JDBC per i precedenti release, consultare Aggiornamenti della V5R2 per il supporto JDBC di IBM Toolbox per Java.

Supporto UTF-8 e UTF-16

I dati UTF-8 vengono memorizzati in un campo di carattere con un CCSID di 1208. Un carattere UTF-8 è un numero variabile di byte (uno, due, tre o quattro) per u carattere non combinato e un qualsiasi numero di byte per un carattere combinato. La lunghezza specificata per un campo di carattere corrisponde al numero massimo di che il campo può contenere. E' possibile inserire una tag nei seguenti tipi di dati con UTF-8 1208 CCSID:

- Carattere a lunghezza fissa (CHAR)
- Carattere a lunghezza variabile (VARCHAR)
- LOB del carattere (CLOB)

I dati UTF-16 vengono memorizzati in un campo grafico con un CCSID di 1200. Un carattere UTF-16 può avere una lunghezza di due o quattro byte (cioè, Surrogato) per un carattere non combinato e un qualsiasi numero di byte per un carattere combinato. La lunghezza specificata per un campo di dati grafico corrisponde al numero massimo di caratteri a due byte che il campo può contenere. E' possibile inserire una tag nei seguenti tipi di dati con UTF-16 1200 CCSID:

- Carattere a lunghezza fissa (GRAPHIC)
- Carattere a lunghezza variabile (VARGRAPHIC)
- LOB di carattere a doppio byte (DBCLOB)

Supporto BINARY e VARBINARY

I tipi di dati BINARY e VARBINARY sono simili ai tipi di dati CHAR e VARCHAR ma contengono dati BINARY piuttosto che dati carattere. I campi BINARY hanno una lunghezza fissa. I campi VARBINARY hanno una lunghezza variabile. I tipi di dati BINARY e VARBINARY dispongono delle seguenti caratteristiche:

- Il CCSID per i tipi BINARY è 65535
- Nell'assegnazione e nel confronto, i tipi di dati BINARY sono compatibili solo con altri tipi di dati BINARY (BINARY, VARBINARY e BLOB)
- Il carattere di riempimento per i tipi di dati BINARY è x'00' piuttosto del carattere vuoto
- Nelle situazioni in cui è necessario eliminare i caratteri finali per evitare troncamenti, vengono eliminati i caratteri x'00' e non i caratteri vuoti finali
- Quando si effettua un confronto dei tipi di dati BINARY, per essere uguali i campi devono avere gli stessi dati e la stesa lunghezza. Gli zeri finali non vengono ignorati durante il confronto

- Quando si effettua un confronto dei tipi di dati BINARY, se due campi hanno lunghezze differenti, il campo più corto viene considerato meno rispetto al campo più lungo nel caso in cui i campi siano uguali a seconda della lunghezza del campo più corto

Supporto IDP (Increased Decimal Precision)
La precisione decimale supporta ora fino a 63 cifre. Sono state aggiunte tre proprietà, ″valore minimo della scala di divisione″, ″valore massimo di precisione″ e ″valore massimo della scala″ e sono stati aggiunti sei metodi a AS400JDBCDataSource, setMinimumDivideScale(int divideScale), getMinimumDivideScale(), setMaximumPrecision(int precision), getMaximumPrecision(), setMaximumScale(int scale) e getMaximumScale(). Il valore minimo della scala di divisione specifica il valore minimo della scala di divisione per il risultato della divisione decimale ed è impostato su un numero da 0 a 9. Il valore massimo di precisione specifica il valore massimo di precisione decimale utilizzato dal database ed è impostato su 31 o 63. Il valore massimo della scale specifica il valore massimo della scala utilizzato dal database ed è impostato su un qualsiasi numero da 0 a 63.

Supporto LOB 2 GB
Gli aggiornamenti apportati al JDBC di IBM Toolbox per Java consentono ora di utilizzare il supporto LOB 2 GB.

Supporto cursore non sensibile
Il supporto cursore supporta ora i cursori non sensibili. Quando si utilizza ResultSet con TYPE_SCROLL_INSENSITIVE, viene utilizzato un cursore non sensibile. ResultSet non mostra le modifiche al sottostante database mentre è aperto.

Supporto MQT (Materialized Query Table)
Restituisce ″MATERIALIZED QUERY TABLE″ come TABLE_TYPE in una chiamata a DatabaseMetaData.getTables().

Funzioni potenziate di JDBC per OS/400 Versione 5 Release 2

Le funzioni potenziate di JDBC per OS/400 Versione 5 Release 2 includono:

- Rimozione della restrizione 'FOR UPDATE'
- Modifica nel troncamento dati
- Richiamo e modifica delle colonne e dei parametri per nome
- Richiamo delle chiavi autogenerate
- Prestazioni potenziate durante l'esecuzione di istruzioni SQLINSERT in un batch
- Supporto potenziato per ResultSet.getRow()
- Supporto potenziato per utilizzare lettere sia maiuscole che minuscole nei nomi di colonne
- Specifica della conservabilità per Statement, CallableStatement e PreparedStatement
- Supporto potenziato all'isolamento della transazione

Rimozione della restrizione 'FOR UPDATE'
Non è più necessario specificare FOR UPDATE sulle istruzioni SELECT per garantirsi un cursore aggiornabile. Durante il collegamento a V5R1 e a versioni più recenti di OS/400, IBM Toolbox per Java rispetta qualunque combinazione trasmessa quando si creano istruzioni. Il valore predefinito continua ad essere un cursore di sola lettura se non viene specificata una combinazione.

Il troncamento dati lancia eccezioni solo quando i dati carattere troncati sono scritti nel database
Le regole del troncamento dei dati per IBM Toolbox per Java ora sono uguali a quelle per l'Unità di controllo JDBC di IBM Developer Kit per Java.Per ulteriori informazioni, consultare Proprietà JDBC di IBM Toolbox per Java.

Richiamo e modifica delle colonne e dei parametri per nome
I nuovi metodi consentono di richiamare ed aggiornare le informazioni per nome colonna in ResultSet e richiamare ed impostare informazioni per nome parametro in CallableStatement. Ad esempio, in ResultSet, dove è stato precedentemente utilizzato quanto segue:

```
ResultSet rs = statement.executeQuery( SELECT * FROM MYCOLLECTION/MYTABLE );
     rs.getString(1);
```

Ora è possibile utilizzare:

```
ResultSet rs = statement.executeQuery( SELECT * FROM MYCOLLECTION/MYTABLE );
     rs.getString( 'STUDENTS' );
```

Si tenga presente che l'accesso ai parametri per indice scaturisce prestazioni migliori rispetto all'accesso per nome. E' possibile anche specificare i nomi dei parametri da impostare in CallableStatement. Dove sarebbe stato possibile utilizzare quanto segue in CallableStatement:

```
CallableStatement cs = connection.prepareCall( CALL MYPGM (?) );
cs.setString( 1 );
```

Ora è possibile utilizzare:

```
CallableStatement cs = connection.prepareCall( CALL MYPGM (?) );
cs.setString( 'PARAM_1' );
```

Per utilizzare questi nuovi metodi, sono necessari JDBC 3.0 o versioni successive e Java 2 Platform, versione 1.4 (Standard o Enterprise Edition).

Richiamo delle chiavi autogenerate

Il metodo getGeneratedKeys() inAS400JDBCStatement richiama ogni chiave autogenerata creata come risultato dell'esecuzione dell'oggetto Statement.Quando l'oggetto Statement non crea alcuna chiave, viene riportato un oggetto ResultSet vuoto. Al momento il server supporta solo una chiave autogenerata (la chiave per l'ultima riga inserita). Il seguente esempio mostra come si può inserire un valore in una tabella e poi richiamare la chiave autogenerata:

```
Statement s =
   statement.executeQuery("INSERT INTO MYSCHOOL/MYSTUDENTS (FIRSTNAME) VALUES ('JOHN'");
     ResultSet rs = s.getGeneratedKeys();
               // Attualmente il server iSeries supporta solo la restituzione di una chiave
               // generata automaticamente -- la chiave dell'ultima riga inserita.
           rs.next ();
     String autoGeneratedKey = rs.getString(1);
               // Usare la chiave generata automaticamente, ad esempio,
               // come la chiave primaria in un'altra tabella
```

Per richiamare le chiavi autogenerate, sono necessari JDBC 3.0 o versioni successive e Java 2 Platform, versione 1.4 (Standard o Enterprise Edition). Il richiamo delle chiavi autogenerate richiede anche il collegamento a V5R2 o alla versione successiva di OS/400.

Prestazioni potenziate durante l'esecuzione di istruzioni SQLINSERT in un batch

Le prestazioni di esecuzione di istruzioni SQLINSERT in un batch sono state potenziate. Eseguire le istruzioni di SQL in un batch utilizzando i differenti metodi addBatch() disponibili in AS400JDBCStatement, AS400JDBCPreparedStatement e AS400JDBCCallableStatement. Il supporto batch aggiornato influenza solo richieste di inserimento. Ad esempio, l'utilizzo del supporto batch per elaborare diversi inserimenti fa sì che solo uno venga inviato al server. Tuttavia, l'utilizzo di un supporto batch per elaborare un inserimento, un aggiornamento ed una cancellazione invia ogni richiesta individualmente.

Per utilizzare il supporto batch, sono necessari JDBC 2.0 o versioni successive e Java 2 Platform, versione 1.2 (Standard o Enterprise Edition).

Supporto potenziato per ResultSet.getRow()

Precedentemente, l'unità di controllo JDBC di IBM Toolbox per Java era stata limitata nel supporto per il metodo getRow() in ResultSet.Nello specifico, l'utilizzo di ResultSet.last(), ResultSet.afterLast() e ResultSet.absolute() con un valore negativo rendeva non disponibile il corrente numero di riga. Le precedenti restrizioni vengono eliminate ed il metodo diviene pienamente funzionale.

Utilizzo di lettere sia maiuscole che minuscole nei nomi delle colonne
I metodi di IBM Toolbox per Java devono far corrispondere tutti i nomi di colonna forniti dall'utente o dall'applicazione ai nomi che si trovano nella tabella del database. In ogni caso, quando un nome di colonna non è racchiuso tra apici, IBM Toolbox per Java lo modifica in caratteri maiuscoli prima di confrontarlo con i nomi sul server. Quando il nome della colonna è racchiuso tra gli apici, esso deve corrispondere esattamente al nome sul server oppure IBM Toolbox per Java invia un'eccezione.

Specifica della conservabilità in Statement, CallableStatement e PreparedStatement create
I nuovi metodi in AS400JDBCConnection consentono di specificare il supporto per Statement, CallableStatement e PreparedStatement create.Il supporto determina se il cursore rimane aperto o chiuso quando si convalida la transazione. E' ora possibile avere un'istruzione che possiede un supporto differente rispetto all'oggetto di collegamento. Inoltre, gli oggetti di collegamento possono avere più oggetti Statement aperti, ognuno con un differente supporto specificato. La chiamata alla convalida fa sì che ogni istruzione venga gestita in accordo con il supporto specificato per quella istruzione.

Il supporto è derivato dal seguente ordine di precedenza:

1. Supporto specificato durante la creazione dell'istruzione utilizzando i metodi createStatement(), prepareCall() o prepareStatement() della classe Connection.
2. Supporto specificato utilizzando Connection.setHoldability(int).
3. Supporto specificato dalla proprietà Congelamento cursore JDBC di IBM Toolbox per Java (quando non sono utilizzati metodi in 1. o 2.)

Per utilizzare questi metodi, sono necessari JDBC 3.0 o versioni successive e Java 2 Platform, versione 1.4 (Standard o Enterprise Edition). Inoltre, i server che eseguono V5R1 o una versione precedente di OS/400 possono utilizzare solo il supporto specificato dalla proprietà Congelamento cursore di JDBC.

Supporto potenziato all'isolamento della transazione
L'unità di controllo JDBC di IBM Toolbox per Java ora presenta il supporto per la commutazione ad un livello di isolamento della transazione di *NONE dopo aver effettuato un collegamento. Prima di V5R2, l'unità di controllo JDBC di IBM Toolbox per Java emetteva un'eccezione quando si passava a *NONE dopo aver stabilito un collegamento.

Proprietà JDBC di IBM Toolbox per Java

E' possibile specificare molte proprietà quando ci si collega ad un database server utilizzando JDBC. Tutte le proprietà sono facoltative ed è possibile specificarle come parte dell'URL o in un oggetto java.util.Properties. Se una proprietà viene impostata sia sull'URL che sull'oggetto Proprietà, sarà utilizzato il valore sull'URL.

Nota: il seguente elenco non include le proprietà DataSource.

Le seguenti tabelle elencano le differenti proprietà di collegamento riconosciute da questo programma di controllo. Alcune proprietà incidono sulle prestazioni, mentre altre sono attributi del lavoro del server. Le tabelle organizzano le proprietà nelle seguenti categorie:

- "Proprietà generali" a pagina 329
- "Proprietà del server" a pagina 329
- "Proprietà del formato" a pagina 332
- "Proprietà delle prestazioni" a pagina 333
- "Proprietà della funzione di ordinamento" a pagina 336
- "Altre proprietà" a pagina 337

Proprietà generali

Le proprietà generali sono attributi del sistema che specificano l'utente, la parola d'ordine e, se necessaria, una richiesta per il collegamento al server.

Proprietà generali	Descrizione	Richiesto	Opzioni	Valore predefinito
″password″	Specifica la parola d'ordine per il collegamento al server. Se non specificato, verrà richiesto all'utente, a meno che la proprietà ″prompt″ non sia impostata su ″false″, in questo caso il tentativo di collegamento avrà esito negativo.	no	parola d'ordine del server	(verrà richiesto all'utente)
″prompt″	Specifica se all'utente vengono richiesti l'ID utente e la parola d'ordine per il collegamento al server. Se è impossibile stabilire un collegamento senza richiesta e se questa proprietà è impostata su ″false″, il tentativo di collegamento avrà esito negativo.	no	″true″ ″false″	″true″
″user″	Specifica il nome utente per il collegamento al server. Se non specificato, verrà richiesto all'utente, a meno che la proprietà ″prompt″ non sia impostata su ″false″, in questo caso il tentativo di collegamento avrà esito negativo.	no	utente del server	(verrà richiesto all'utente)

Proprietà del server

Le proprietà del server specificano gli attributi che regolano le transazioni, le librerie e i database.

Proprietà del server	Descrizione	Richiesto	Opzioni	Valore predefinito
″cursor hold″	Specifica se congelare il cursore attraverso le transazioni. Se questa proprietà è impostata su ″true″, i cursori non sono chiusi quando una transazione viene sottoposta a commit o ripristinata allo stato precedente. Tutte le risorse acquisite durante l'unità di lavoro vengono congelate, ma i blocchi su righe specifiche e oggetti implicitamente acquisiti durante l'unità del lavoro vengono rilasciati.	no	″true″″false″	″true″
″cursor sensitivity″	Specifica la sensibilità del cursore richiesta dal database. La funzionalità dipende da resultSetType: • ResultSet.TYPE_FORWARD_ONLY o ResultSet.TYPE_SCROLL_SENSITIVE indicano che il valore di questa proprietà controlla la sensibilità del programma Java richiesta dal database. • ResultSet.TYPE_SCROLL_INSENSITIVE fa in modo che questa proprietà venga ignorata. Questa proprietà viene ignorata quando ci si collega a sistemi su cui è in esecuzione la versione V5R1 e versioni precedenti di OS/400.	no	″″ (Utilizzare il tipo ResultSet per stabilire la sensibilità del cursore) ″asensitive″ ″sensitive″ ″insensitive″	″″

Proprietà del server	Descrizione	Richiesto	Opzioni	Valore predefinito
″database name″	Specifica il database da utilizzare per il collegamento, compreso quello memorizzato in un IASP (Independent Auxiliary Storage Pool).Questa proprietà è valida solo quando ci si collega ad un V5R2 o ad una versione successiva di OS/400. Quando si specifica il nome del database, è necessario che il nome esista nell'indirizzario del database relazionale sul server. I seguenti criteri determinano il database a cui si accede: • Quando questa proprietà viene utilizzata per specificare un database, si utilizza il database specificato. Quando il database specificato non esiste, il collegamento ha esito negativo. • Quando questa proprietà viene utilizzata per specificare *SYSBAS come nome del database, si utilizza il database predefinito del sistema. • Quando questa proprietà viene omessa, si utilizza il nome del database specificato nella descrizione del lavoro per il profilo utente. Quando la descrizione del lavoro non specifica un nome del database, si utilizza il database predefinito del sistema.	no	Nome database ″*SYSBAS″	Si utilizza il nome del database specificato nella descrizione del lavoro per il profilo utente. Quando la descrizione del lavoro non specifica un nome del database, si utilizza il database predefinito del sistema.

Proprietà del server	Descrizione	Richiesto	Opzioni	Valore predefinito
″libraries″	Specifica una o più librerie che si desidera aggiungere o sostituire nell'elenco librerie del lavoro server e facoltativamente imposta la libreria predefinita (schema predefinito). **Elenco librerie** Il server utilizza le librerie specificate per definire nomi procedura memorizzati non qualificati e le procedure memorizzate li utilizzano per definire nomi non qualificati. Per specificare più librerie, utilizzare le virgole o gli spazi per separare le singole voci. E' possibile utilizzare *LIBL come placeholder per l'elenco librerie corrente del lavoro server: • Quando la prima voce è *LIBL, le librerie specificate vengono aggiunte all'elenco librerie corrente del lavoro server • Quando non viene utilizzata la voce *LIBL, le librerie specificate sostituiscono l'elenco librerie corrente del lavoro server Per ulteriori informazioni sulle proprietà dell'elenco librerie, consultare Proprietà JDBC LibraryList. **Schema predefinito** Il server utilizza lo schema predefinito per definire i nomi non qualificati nelle istruzioni SQL. Ad esempio, nell'istruzione ″SELECT * FROM MYTABLE″, il server consulta solo lo schema predefinito di MYTABLE. Lo schema predefinito può essere specificato sull'URL del collegamento. Quando ciò non viene effettuati, verranno applicate le condizioni riportate di seguito, a seconda del fatto che si stia utilizzando la denominazione SQL o la denominazione di sistema. • **Denominazione SQL** Quando non si specifica lo schema predefinito sull'URL del collegamento: – La prima voce (a meno che non sia *LIBL) diventa lo schema predefinito – Quando la prima voce è *LIBL, la seconda voce diventa lo schema predefinito – Quando non si imposta questa proprietà oppure quando contiene solo *LIBL, il profilo utente diventa lo schema predefinito • **Denominazione sistema** Quando non si specifica lo schema predefinito sull'URL del collegamento: – Non viene impostato alcuno schema predefinito e il server utilizza le librerie specificate per la ricerca dei nomi non qualificati – Quando questa proprietà non viene impostata o quando contiene solo *LIBL, il server utilizza l'elenco librerie corrente del lavoro server per la ricerca dei nomi non qualificati	no	Elenco di librerie server, separate da virgole o spazi	″*LIBL″
″maximum precision″	Specifica il valore massimo della precisione decimale che dovrebbe utilizzare il database.	no	″31″ ″63″	″31″
″maximum scale″	Specifica il valore massimo della scale che dovrebbe utilizzare il database.	no	″0″-″63″	″31″

Proprietà del server	Descrizione	Richiesto	Opzioni	Valore predefinito
″minimum divide scale″	Specifica il valore minimo della scala per il risultato della divisione decimale.	no	″0″ ″1″ ″2″ ″3″ ″4″ ″5″ ″6″ ″7″ ″8″ ″9″	″0″
″package ccsid″	Specifica la codifica carattere da utilizzare per il pacchetto SQL e le istruzioni inviate al server.	no	″1200″ (UCS-2) ″13488″ (UTF-16)	″13488″
″transaction isolation″	Specifica l'isolamento transazione predefinito.	no	″none″ ″read uncommitted″ ″read committed″ ″repeatable read″ ″serializable″	″read uncommitted″
″translate hex″	Specifica in che modo vengono interpretate le costanti letterali esadecimali.	no	″character″ (Interpretare le costanti letterali esadecimali come dati carattere) ″binary″ (Interpretare le costanti letterali esadecimali come dati BINARY)	″character″
″true autocommit″	Specifica se il collegamento deve utilizzare supporto true autocommit. True autocommit indica che l'autocommit è attivo ed è in esecuzione con un livello di isolamento diverso da *NONE. Per impostazione predefinita, il programma di controllo gestisce l'autocommit eseguendolo con il livello di isolamento server *NONE.	no	″true″ (Utilizzare true autocommit.) ″false″ (Non utilizzare true autocommit.)	″false″

Proprietà del formato

Le proprietà del formato specificano i formati della data e dell'ora, i separatori della data e decimali e la tabella che denomina le convenzioni utilizzate all'interno delle istruzioni SQL.

Proprietà formato	Descrizione	Richiesto	Opzioni	Valore predefinito
″date format″	Specifica il formato della data utilizzato nelle costanti letterali della data all'interno delle istruzioni SQL.	no	″mdy″ ″dmy″ ″ymd″ ″usa″ ″iso″ ″eur″ ″jis″ ″julian″	(lavoro del server)
″date separator″	Specifica il separatore di data utilizzato nelle costanti letterali della data all'interno delle istruzioni SQL. Questa proprietà non ha alcun effetto a meno che la proprietà ″date format″ non sia impostata su ″julian″, ″mdy″, ″dmy″ o ″ymd″.	no	″/″ (barra) ″-″ (trattino) ″.″ (punto) ″,″ (virgola) ″b″ (spazio)	(lavoro del server)
″decimal separator″	Specifica il separatore decimale utilizzato in costanti letterali numeriche all'interno delle istruzioni SQL.	no	″.″ (punto) ″,″ (virgola)	(lavoro del server)
″naming″	Specifica la convenzione di denominazione utilizzata quando ci si riferisce a tabelle.	no	″sql″ (come in `schema.table`) ″system″ (come in `schema/tabella`)	″sql″
″time format″	Specifica il formato dell'ora utilizzato nelle costanti letterali dell'ora all'interno delle istruzioni SQL.	no	″hms″ ″usa″ ″iso″ ″eur″ ″jis″	(lavoro del server)
″time separator″	Specifica il separatore dell'ora utilizzato nelle costanti letterali dell'ora all'interno delle istruzioni SQL. Questa proprietà non ha alcun effetto a meno che la proprietà ″time format″ non sia impostata su ″hms″.	no	″:″ (due punti) ″.″ (punto) ″,″ (virgola) ″b″ (spazio)	(lavoro del server)

Proprietà delle prestazioni

Le proprietà delle prestazioni sono attributi che includono la memorizzazione nella cache, la conversione dei dati, la compressione dei dati e la messa in corrispondenza che influisce sulle prestazioni.

Proprietà prestazioni	Descrizione	Richiesto	Opzioni	Valore predefinito
″big decimal″	Specifica se un oggetto intermedio java.math.BigDecimal viene utilizzato per conversioni decimali compresse e a zonatura decimale. Se questa proprietà è impostata su ″true″, un oggetto intermedio java.math.BigDecimal viene utilizzato per conversioni decimali compresse e a zonatura come descritto nella specifica JDBC. Se questa proprietà è impostata su ″false″, non vengono utilizzati oggetti intermedi per conversioni compresse e a zonatura decimale. Tali valori sono invece convertiti direttamente in e da valori doppi Java. Tali conversioni saranno più veloci ma è possibile che non seguano tutte le regole di conversione e di troncamento dei dati documentate dalla specifica JDBC.	no	″true″ ″false″	″true″
″block criteria″	Specifica i criteri per richiamare i dati dal server in blocchi di record. L'indicazione di un valore non-zero per questa proprietà ridurrà la frequenza di comunicazione con il server, quindi, migliorerà le prestazioni. Assicurarsi che il blocco record sia disattivo se il cursore deve essere utilizzato per successivi AGGIORNAMENTI, altrimenti la riga aggiornata non sarà necessariamente la riga corrente.	no	″0″ (nessun blocco record) ″1″ (blocco se viene specificato FOR FETCH ONLY) ″2″ (blocco a meno che non sia specificato FOR UPDATE)	″2″
″block size″	Specifica la dimensione del blocco (in kilobyte) da richiamare dal server e da memorizzare nella cache del client. Questa proprietà non ha alcun effetto a meno che la proprietà ″block criteria″ non sia non-zero. Dimensioni maggiori del blocco riducono la frequenza di comunicazioni con il server, quindi possono migliorare le prestazioni.	no	″0″ ″8″ ″16″ ″32″ ″64″ ″128″ ″256″ ″512″	″32″
″data compression″	Specifica se i dati della serie di risultati vengono compressi. Se questa proprietà è impostata su ″true″, i dati della serie di risultati vengono compressi. Se questa proprietà è impostata su ″false″, i dati della serie di risultati non vengono compressi. La compressione dei dati può migliorare le prestazioni quando si richiamano serie di risultati estese.	no	″true″ ″false″	″true″

Proprietà prestazioni	Descrizione	Richiesto	Opzioni	Valore predefinito
″extended dynamic″	Specifica se utilizzare un supporto dinamico esteso. Un supporto dinamico esteso fornisce un meccanismo per memorizzare nella cache le istruzioni dinamiche SQL sul server. La prima volta che viene preparata un'istruzione SQL speciale, viene memorizzata in un pacchetto SQL nel server. Se il pacchetto non esiste, viene creato automaticamente. Nelle preparazioni successive della stessa istruzione SQL, è possibile che il server ignori una parte significativa dell'elaborazione utilizzando le informazioni memorizzate nel pacchetto SQL. Se questo è impostato su ″true″, è necessario impostare un nome pacchetto utilizzando la proprietà ″package″.	no	″true″ ″false″	″false″
″lazy close″	Specifica se ritardare la chiusura dei cursori fino a richieste successive. Questo migliorerà le prestazioni complessive riducendo il numero totale delle richieste.	no	″true″″false″	″false″
″lob threshold″	Specifica la massima dimensione (in byte) del LOB (large object) che è possibile richiamare come parte di una serie di risultati. I LOB più grandi di tale soglia saranno richiamati in gruppi, utilizzando una comunicazione straordinaria con il server. Le soglie LOB più estese ridurranno la frequenza di comunicazioni con il server, ma scaricheranno più dati LOB, anche se non utilizzati. Le soglie LOB più esigue possono aumentare la frequenza di comunicazioni con il server, ma scaricheranno solo i dati LOB necessari.	no	″0″ - ″16777216″	″32768″
″package″	Specifica il nome base del pacchetto SQL. Tenere presente che solo i primi sette caratteri vengono utilizzati per creare il nome del pacchetto SQL sul server. Questa proprietà non ha alcun effetto a meno che la proprietà ″extended dynamic″ non sia impostata su ″true″. Inoltre, è necessario impostare questa proprietà se la proprietà ″extended dynamic″ è impostata su ″true″.	no	pacchetto SQL	″″
″package add″	Specifica se aggiungere nuove istruzioni al pacchetto SQL specificato nella proprietà ″package″. Questa proprietà non ha alcun effetto a meno che la proprietà ″extended dynamic″ non sia impostata su ″true″.	no	″true″ ″false″	″true″

Proprietà prestazioni	Descrizione	Richiesto	Opzioni	Valore predefinito
″package cache″	Specifica se memorizzare nella cache una sottoserie di informazioni del pacchetto SQL nella memoria del client. La memorizzazione nella cache dei pacchetti SQL riduce localmente le comunicazioni con il server per le preparazioni e le descrizioni. Questa proprietà non ha alcun effetto a meno che la proprietà ″extended dynamic″ non sia impostata su ″true″.	no	″true″ ″false″	″false″
″package criteria″	Specifica il tipo di istruzioni SQL da memorizzare nel pacchetto SQL. Ciò può essere utile per migliorare le prestazioni delle condizioni di collegamento complesse. Questa proprietà non ha alcun effetto a meno che la proprietà ″extended dynamic″ non sia impostata su ″true″.	no	″default″ (memorizza solo le istruzioni SQL con i contrassegni del parametro nel pacchetto) ″select″ (memorizza tutte le istruzioni SQL SELECT nel pacchetto)	″default″
″package error″	Specifica come agire nel caso si verifichino errori del pacchetto SQL. Quando si verifica un errore del pacchetto SQL, il programma di controllo avvierà facoltativamente una SQLException o invierà un messaggio di avvertenza al collegamento, in base al valore di tale proprietà. Questa proprietà non ha alcun effetto a meno che la proprietà ″extended dynamic″ non sia impostata su ″true″.	no	″exception″ ″warning″ ″none″	″warning″
″package library″	Specifica la libreria per il pacchetto SQL. Questa proprietà non ha alcun effetto a meno che la proprietà ″extended dynamic″ non sia impostata su ″true″.	no	Libreria per il pacchetto SQL	″QGPL″
″prefetch″	Specifica se mettere in corrispondenza i dati eseguendo un'istruzione SELECT. Ciò aumenterà le prestazioni durante l'accesso alle righe iniziali in ResultSet.	no	″true″ ″false″	″true″
″qaqqinilib″	Specifica un nome libreria QAQQINI. Utilizzato per specificare la libreria che contiene il file qaqqini da utilizzare. Un file qaqqini contiene tutti gli attributi che potrebbero influenzare le prestazioni del motore database DB2 UDB per iSeries.	no	″Nome libreria QAQQINI″	(valore predefinito del server)

Proprietà della funzione di ordinamento

Le proprietà della funzione di ordinamento specificano come il server effettua la memorizzazione e gli ordinamenti.

Proprietà funzione ordinamento	Descrizione	Richiesto	Opzioni	Valore predefinito
″sort″	Specifica il modo in cui il server ordina i record prima di inviarli al client.	no	″hex″ (esegue l'ordinamento in base ai valori esadecimali) ″job″ (esegue l'ordinamento in base all'impostazione per il lavoro server) ″language″ (esegue l'ordinamento in base alla lingua impostata nella proprietà ″sort language″) ″table″ (esegue l'ordinamento in base alla tabella sequenza di ordinamento impostata nella proprietà ″sort table″)	″job″
″sort language″	Specifica un ID della lingua di tre caratteri da utilizzare per la selezione di una sequenza di ordinamento. Questa proprietà non ha alcun effetto a meno che la proprietà ″sort″ non sia impostata su ″language″.	no	Id lingua	ENU
″sort table″	Specifica la libreria e il nome del file di una tabella sequenza ordinamento memorizzata sul server. Questa proprietà non ha alcun effetto a meno che la proprietà ″sort″ non sia impostata su ″table″.	no	Nome tabella ordinamento completo	″″
″sort weight″	Specifica come il server considera il carattere mentre ordina i record. Questa proprietà non ha alcun effetto a meno che la proprietà ″sort″ non sia impostata su ″language″.	no	″shared″ (caratteri maiuscoli e minuscoli ordinati per lo stesso carattere) ″unique″ (caratteri maiuscoli e minuscoli ordinati per caratteri differenti)	″shared″

Altre proprietà

Non è possibile suddividere per categorie le altre proprietà. Esse determinano quale unità di controllo JDBC viene utilizzata e specifica le opzioni correlate al livello di accesso del database, al tipo di stringa bidirezionale, al troncamento dei dati eccetera.

Altra proprietà	Descrizione	Richiesto	Opzioni	Valore predefinito
″access″	Specifica il livello di accesso al database per il collegamento.	no	″all″ (sono consentite tutte le istruzioni SQL) ″read call″ (istruzioni SELECT e CALL consentite) ″read only″ (solo istruzioni SELECT consentite)	″all″
″behavior override″	Specifica quali funzionalità dell'unità di controllo JDBC di IBM Toolbox per Java sostituire. E' possibile modificare più funzionalità insieme aggiungendo le costanti e inoltrando tale somma a questa proprietà. Assicurarsi che l'applicazione gestisca correttamente la funzionalità modificata.	no	″″ (non sovrascrivere nessuna funzione) ″1″ (non emettere un'eccezione ma al contrario restituire un valore null per la serie di risultati se Statement.executeQuery() o PreparedStatement.executeQuery() non restituisce una serie di risultati)	″″
″bidi string type″	Specifica il tipo di stringa di emissione dei dati bidirezionali. Consultare BidiStringType per ulteriori informazioni.	no	″″ (utilizzare CCSID per stabilire il tipo di stringa bidirezionale) ″0″ (il tipo di stringa predefinita per dati non bidirezionali (LTR)) ″4″ ″5″ ″6″ ″7″ ″8″ ″9″ ″10″ ″11″	″″
″bidi implicit reordering″	Specifica se deve essere utilizzato il riordino LTR-RTL bidi implicito.	no	″true″″false″	″true″
″bidi numeric ordering″	Specifica se deve essere utilizzata la funzione di numeric ordering round trip.	no	″true″ ″false″	″false″
″data truncation″	Specifica se il troncamento dei **dati carattere** crea messaggi di avvertenza ed eccezioni. Quando questa proprietà è ″true″, è necessario considerare quanto segue: • Se si scrivono i dati carattere troncati sul database viene inviata un'eccezione • Se si utilizzano dati carattere troncati in un'interrogazione viene inviato un messaggio di avvertenza. Quando questa proprietà è ″false″, la scrittura di dati troncati sul database o l'utilizzo di tali dati in un'interrogazione non crea né un'eccezione né un messaggio di avvertenza. Il valore predefinito è ″true″. Questa proprietà non influenza i **dati numerici**. Scrivendo dati numerici troncati sul database viene lanciato sempre un errore, mentre utilizzando dati numerici troncati in un'interrogazione, viene prodotto un messaggio di avvertenza.	no	″true″″false″	″true″

Altra proprietà	Descrizione	Richiesto	Opzioni	Valore predefinito
″driver″	Specifica l'implementazione dell'unità di controllo JDBC. L'unità di controllo JDBC IBM Toolbox per Java può utilizzare differenti implementazioni dell'unità di controllo JDBC basate sull'ambiente. Se l'ambiente è una JVM iSeries sullo stesso server del database a cui il programma si sta collegando, è possibile utilizzare l'unità di controllo JDBC di IBM Developer Kit per Java. In tutti gli altri ambienti, si utilizza l'unità di controllo JDBC di IBM Toolbox per Java. Questa proprietà non ha alcun effetto se è impostata la proprietà ″secondary URL″.	no	″toolbox″ (utilizzare solo l'unità di controllo JDBC IBM Toolbox per Java). ″native″ (utilizzare l'unità di controllo JDBC di IBM Developer Kit per Java se è in esecuzione sul server, altrimenti utilizzare l'unità di controllo JDBC di IBM Toolbox per Java).	″toolbox″
″errors″	Specifica il numero dei dettagli da riportare nel messaggio per gli errori che si verificano sul server.	no	″basic″ ″full″	″basic″
″extended metadata″	Specifica se il programma di controllo richiede metadati estesi dal server. Impostare questa proprietà su True aumenta la precisione delle informazioni restituite dai seguenti metodi ResultSetMetaData: • getColumnLabel(int) • isReadOnly(int) • isSearchable(int) • isWriteable(int) Inoltre, impostare questa proprietà su true abilita il supporto per il metodo ResultSetMetaData.getSchemaName(int). L'impostazione di questa proprietà su ″true″ potrebbe rallentare le prestazioni perché richiede che venga richiamato un maggior numero di informazioni dal server. Lasciare la proprietà come valore predefinito (false) a meno che non siano necessarie informazioni più specifiche dai metodi elencati. Ad esempio, quando questa proprietà è disattiva (false), ResultSetMetaData.isSearchable(int) ritorna sempre su ″true″ poiché il programma di controllo non dispone di informazioni sufficienti dal server per dare un giudizio. L'attivazione di questa proprietà (true) impone al programma di controllo di ottenere i dati corretti dal server. E' possibile utilizzare i metadati estesi solo quando ci si collega ad un server che esegue OS/400 V5R2 o successivo.	no	″true″ ″false″	″false″

Altra proprietà	Descrizione	Richiesto	Opzioni	Valore predefinito
″full open″	Specifica se il server apre completamente un file per ogni interrogazione. Con il valore predefinito, il server ottimizza le richieste aperte. Tale ottimizzazione migliora le prestazioni ma può fallire se il monitor di un database è attivo quando si esegue un'interrogazione più di una volta. Impostare la proprietà su true solo quando vengono emesse delle interrogazioni identiche quando i monitor sono attivi.	no	″true″ ″false″	″false″
″hold input locators″	Specifica se i programma di ricerca di immissione devono essere assegnati come programmi di ricerca che conservano o meno i dati. Se tali programmi conservano i dati, non verranno rilasciati una volta completato il commit.	no	″true″ (tipo conservato) ″false″	″true″
″hold statements″	Specifica se le istruzioni devono rimanere aperte fino a quando non viene raggiunto il limite della transazione quando l'autocommit è inattivo e sono associate ad un programma di ricerca LOB. Per impostazione predefinita, tutte le risorse associate ad un'istruzione vengono rilasciate nel momento in cui l'istruzione viene chiusa. Impostare tale proprietà su true solo quando è necessario accedere ad un programma di ricerca LOB una volta chiusa un'istruzione.	no	″true″″false″	″false″
″key ring name″	Specifica il nome della classe keyring per i collegamenti SSL con il server. Questa proprietà non ha alcun effetto a meno che ″secure″ non sia impostato su true e una parola d'ordine keyring non venga impostata con la proprietà ″key ring password″.	no	″key ring name″	″″
″key ring password″	Specifica la parola chiave per la classe keyring utilizzata per le comunicazioni SSL con il server. Questa proprietà non ha alcun effetto a meno che ″secure″ non sia impostato su true e un nome keyring non venga impostato utilizzando la proprietà ″key ring name″.	no	″key ring password″	″″

Altra proprietà	Descrizione	Richiesto	Opzioni	Valore predefinito
″proxy server″	Specifica il nome host e la porta della macchina della schiera centrale dove il server proxy è in esecuzione. Il formato per tale operazione è *hostname*[:*port*], in cui la porta è facoltativa. Se non è impostato, l'hostname e la porta vengono richiamati dalla proprietà *com.ibm.as400.access.AS400.proxyServer*. La porta predefinita è *3470* (se il collegamento utilizza SSL, la porta predefinita è *3471*). E' necessario che il server proxy sia in esecuzione sulla macchina della schiera centrale. Il nome della macchina della schiera centrale viene ignorato in un ambiente a due schiere.	no	Porta e nome dell'host server proxy	(valore della proprietà server proxy, solo se è impostata)
″remarks″	Specifica il sorgente del testo per le colonne REMARKS in ResultSets restituite dai metodi DatabaseMetaData.	no	″sql″ (commento oggetto SQL) ″system″ (descrizione oggetto OS/400)	″system″
″save password when serialized″	Specifica se salvare la parola d'ordine localmente insieme al resto della proprietà quando questo oggetto origine dati viene serializzato. Il salvataggio della parola d'ordine indica che l'applicazione deve proteggere il formato serializzato dell'oggetto, in quanto l'oggetto contiene tutte le informazioni necessarie per collegarsi al server. Prima di impostare questa proprietà su ″true″, considerare il rischio di sicurezza causato dal salvataggio della parola d'ordine con il resto delle proprietà.	no	″true″″false″	″false″
″secondary URL″	Specifica l'URL da utilizzare per un collegamento sul DriverManager della schiera centrale in un ambiente a schiere multiple, se è diverso da quello già specificato. Questa proprietà consente di utilizzare questo programma di controllo per il collegamento a database differenti dal server iSeries. Utilizzare una barra retroversa come un carattere di uscita prima delle barre retroverse e dei due punti sull'URL.	no	URL JDBC	(URL JDBC corrente)
″secure″	Specifica se un collegamento SSL (Secure Sockets Layer) viene utilizzato per comunicare con il server. I collegamenti SSL sono disponibili solo quando ci si collega a server V4R4 o successivi.	no	″true″ (codificare tutte le comunicazioni client/server) ″false″ (codificare solo la parola d'ordine)	″false″

Altra proprietà	Descrizione	Richiesto	Opzioni	Valore predefinito
″server trace″	Specifica il livello di traccia del lavoro del server JDBC. Quando la traccia è abilitata, si avvia quando il cliente si collega al server e termina quando viene chiuso il collegamento. E′ necessario avviare la traccia prima di collegarsi al server, poiché il cliente abilita il server ad effettuare la traccia solo al momento del collegamento.	no	″0″ (la traccia non è attiva) ″2″ (avviare il monitor del database sul lavoro server JDBC) ″4″ (avviare il debug sul lavoro server JDBC) ″8″ (salvare la registrazione lavoro quando il lavoro server JDBC termina) ″16″ (avviare la traccia lavoro sul lavoro server JDBC) ″32″ (salvare le informazioni SQL) E′ possibile avviare diversi tipi di traccia aggiungendo questi valori insieme. Ad esempio, ″6″ avvia il monitor del database e avvia il debug.	″0″
″thread used″	Specifica se i sottoprocessi vengono utilizzati nella comunicazione con i server host.	no	″true″ ″false″	″true″

Altra proprietà	Descrizione	Richiesto	Opzioni	Valore predefinito
″toolbox trace″	Specifica quale categoria di una traccia IBM Toolbox per Java registrare. I messaggi traccia sono utili per i programmi di debug che richiamano JDBC. Tuttavia, è possibile che si verifichi un'anomalia delle prestazioni con la registrazione dei messaggi di traccia, pertanto, questa proprietà viene impostata solo per il debug. I messaggi traccia vengono registrati su System.out.	no	″″ ″none″ ″datastream″ (registrare il flusso dati tra l'host locale e il sistema remoto) ″diagnostic″ (registrare le informazioni sullo stato dell'oggetto) ″error″ (registrare gli errori che provocano un'eccezione) ″information″ (utilizzate per tracciare il flusso di controllo nel codice) ″warning″ (registrare gli errori recuperabili) ″conversion″ (registrare le conversioni della serie di caratteri tra Unicode e codepage native) ″proxy″ (registrare il flusso di dati tra il client e il server proxy) ″pcml″ (utilizzato per stabilire in che modo PCML interpreta i dati inviati al e dal server) ″jdbc″ (registrare le informazioni jdbc) ″all″ (registrare tutte le categorie) ″thread″ (registrare le informazioni sul sottoprocesso)	″″
″trace″	Specifica se i messaggi traccia vengono registrati. I messaggi traccia sono utili per i programmi di debug che richiamano JDBC. Tuttavia, è possibile che si presenti un'anomalia nelle prestazioni con la registrazione dei messaggi traccia, pertanto, questa proprietà viene impostato solo su ″true″ per il debug. I messaggi traccia vengono registrati su System.out.	no	″true″ ″false″	″false″

Altra proprietà	Descrizione	Richiesto	Opzioni	Valore predefinito
″translate binary″	Specifica se i dati binario sono convertiti. Se questa proprietà viene impostata su ″true″, i campi BINARY e VARBINARY vengono considerati come campi CHAR e VARCHAR.	no	″true″ ″false″	″false″

Proprietà JDBC Librarylist

La proprietà JDBC LibraryList specifica una o più librerie che si desidera aggiungere o sostituire nell'elenco librerie del lavoro server e facoltativamente imposta la libreria predefinita (schema predefinito).

Gli esempi di seguito riportati mostrano quanto menzionato precedentemente:

- Una libreria denominata MYLIBDAW contiene MYFILE_DAW
- Si sta eseguendo questa istruzione SQL:

```
"SELECT * FROM MYFILE_DAW"
```

Scenario	Denominazione SQL	Denominazione di sistema
Regole di base	Viene ricercata sola una libreria. • Se viene specificata una libreria nell'URL, quest'ultima viene utilizzata. Tale libreria diviene quella predefinita. • Se non viene specificata alcuna libreria nell'URL, viene utilizzata la prima libreria nella proprietà 'librerie'. Tale libreria diviene quella predefinita. • Se non sono presenti librerie nell'URL e se no viene specificata alcuna proprietà delle librerie, viene utilizzata la libreria con lo stesso nome del profilo utente che si è collegato. L'elenco librerie del lavoro viene aggiornato con le librerie nella proprietà delle librerie. Ciò potrebbe influenzare la funzione di alcuni trigger e procedure memorizzate. Non influenza nomi non qualificati nelle istruzioni.	L'elenco librerie del lavoro viene aggiornato con le librerie nella proprietà delle librerie. Se viene specificata una libreria predefinita nell'URL, tale libreria sarà quella predefinita.
1. Nessuna libreria specificata.	Lo schema predefinito è il nome del profilo utente.	Nessuno schema predefinito. Viene ricercato l'elenco librerie del lavoro.
2. Libreria predefinita specificata nell'URL.	Lo schema predefinito è la libreria specificata.	Lo schema predefinito è la libreria specificata. L'elenco librerie non viene ricercato per risolvere un nome non qualificato nelle istruzioni SQL.
3. Libreria predefinita specificata tramite la proprietà.	Lo schema predefinito è la libreria specificata.	Nessuno schema predefinito. Vengono ricercate tutte le librerie presenti nell'elenco.
4. Librerie predefinita specificata nell'URL e proprietà.	Lo schema predefinito è la librerie specificata nell'URL. L'elenco librerie viene ignorato.	Lo schema predefinito è la librerie specificata nell'URL. L'elenco librerie non viene ricercato per risolvere un nome non qualificato nelle istruzioni SQL.

Scenario	Denominazione SQL	Denominazione di sistema
5. Proprietà librerie specificata, nome libreria errato	Lo schema predefinito è la libreria specificata	Nessuno schema predefinito. L'elenco Librerie non può essere modificato perché una delle librerie nell'elenco non è stata rilevata, quindi viene utilizzato l'elenco librerie del lavoro.
6. Nessuna libreria nell'URL, specificata la proprietà libreria, file rilevato nella seconda libreria nell'elenco	Lo schema predefinito è la prima libreria nell'elenco, tutte le altre librerie vengono ignorate.	Se sono presenti tutte le librerie, quindi non è presente alcuno schema predefinito, vengono ricercate tutte le librerie nell'elenco, l'elenco sostituisce l'elenco librerie del lavoro. Se una delle librerie nell'elenco non esiste, l'elenco librerie del lavoro non viene modificato.
7. Proprietà libreria specificata, l'elenco inizia con una virgola	Lo schema predefinito è il profilo utente	Nessuno schema predefinito, vengono ricercate tutte le librerie nell'elenco, l'elenco sostituisce l'elenco librerie del lavoro.
8. Proprietà libreria specificata, l'elenco inizia con *LIBL	Lo schema predefinito è il profilo utente	Nessuno schema predefinito, vengono ricercate tutte le librerie nell'elenco, le librerie specificate vengono aggiunte alla fine dell'elenco
9. Proprietà libreria specificata, l'elenco termina con *LIBL	Lo schema predefinito è la prima libreria nell'elenco, il resto dell'elenco viene ignorato	Nessuno schema predefinito, vengono ricercate tutte le librerie nell'elenco, le librerie specificate vengono aggiunte all'inizio dell'elenco librerie del lavoro
10. Libreria URL non valida	Nessuno schema predefinito, viene utilizzato il profilo utente	Nessuno schema predefinito, viene utilizzato l'elenco librerie del lavoro

Nota: quando si specifica uno schema predefinito nell'URL e la proprietà librerie non viene utilizzata, lo schema predefinito viene accodato prima dell'elenco librerie corrente

Tipi SQL JDBC

Tipi SQL non supportati

Non tutti i tipi SQL descritti dalla specifica JDBC sono supportati da DB2 per OS/400. Nei casi in cui un tipo SQL non sia supportato, l'unità di controllo JDBC sostituisce un tipo SQL simile.

La tabella seguente elenca i tipi SQL che non sono supportati e il tipo SQL che l'unità di controllo JDBC sostituisce per ognuno.

Tipi di SQL non supportati	Tipo di SQL sostituito
BIT	SMALLINT
TINYINT	SMALLINT
BIGINT (su OS/400 Versione 4 Release 4 e precedenti)	INTEGER
LONGVARCHAR	VARCHAR
LONGVARBINARY	VARBINARY

Nota: BIGINT è supportato su OS/400 V4R5 e successive versioni.

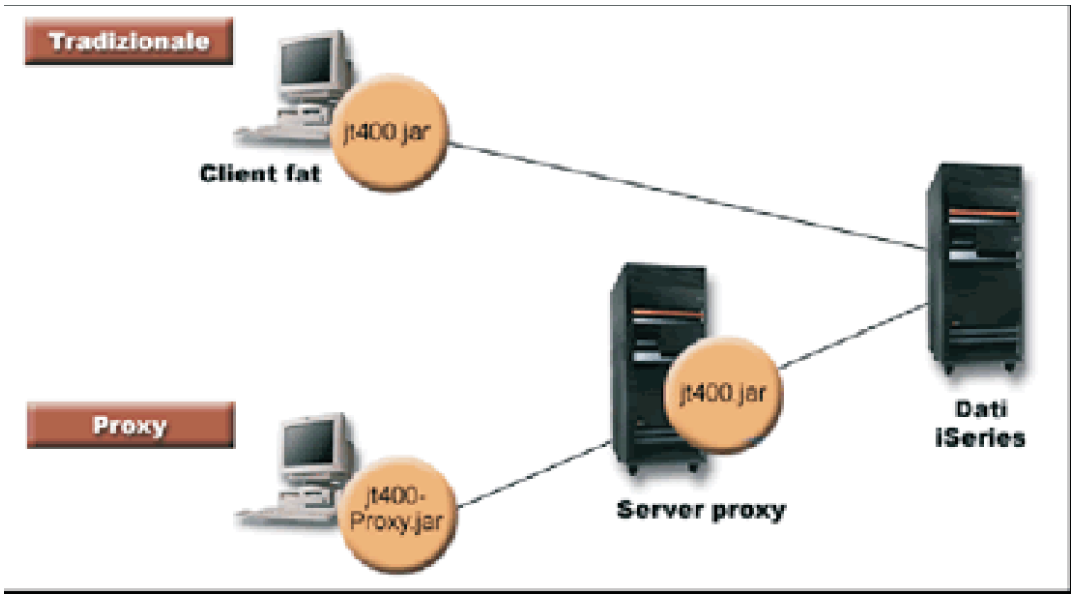
Supporto proxy

L'IBM Toolbox per Java include il supporto proxy per alcune classi. Il supporto proxy è l'elaborazione necessaria all'IBM Toolbox per Java per l'esecuzione di un'attività su una JVM (Java virtual machine) quando l'applicazione si trova su una JVM diversa. Il supporto proxy include l'utilizzo del protocollo SSL (Secure Sockets Layer) per la codifica dei dati.

Le classi proxy si trovano in jt400Proxy.jar, che viene consegnato con il resto dell'IBM Toolbox per Java. Le classi proxy, come le altre classi contenute nell'IBM Toolbox per Java, comprendono una serie di classi Java indipendenti dalla piattaforma che è possibile eseguire su qualsiasi computer che disponga di una Java virtual machine.Le classi proxy smistano tutte le chiamate ai metodi ad un'applicazione del server o a un server proxy. Le classi complete dell'IBM Toolbox per Java si trovano nel server proxy. Quando un client utilizza una classe proxy, la richiesta viene trasferita al server proxy che crea ed amministra gli effettivi oggetti dell'IBM Toolbox per Java.

La Figura 1 mostra in che modo il client standard ed il client proxy si collegano al server. Il server proxy può essere l'iSeries contenente i dati.

Figura 1: modalità di collegamento al server di un client standard e di un client proxy



Un'applicazione che utilizzi il supporto proxy ha prestazioni rallentate rispetto all'utilizzo di classi standard dell'IBM Toolbox per Java, a causa dell'ulteriore comunicazione necessaria per supportare le classi proxy di minori dimensioni. Le applicazioni che effettuano meno chiamate ai metodi subiscono una riduzione inferiore delle prestazioni.

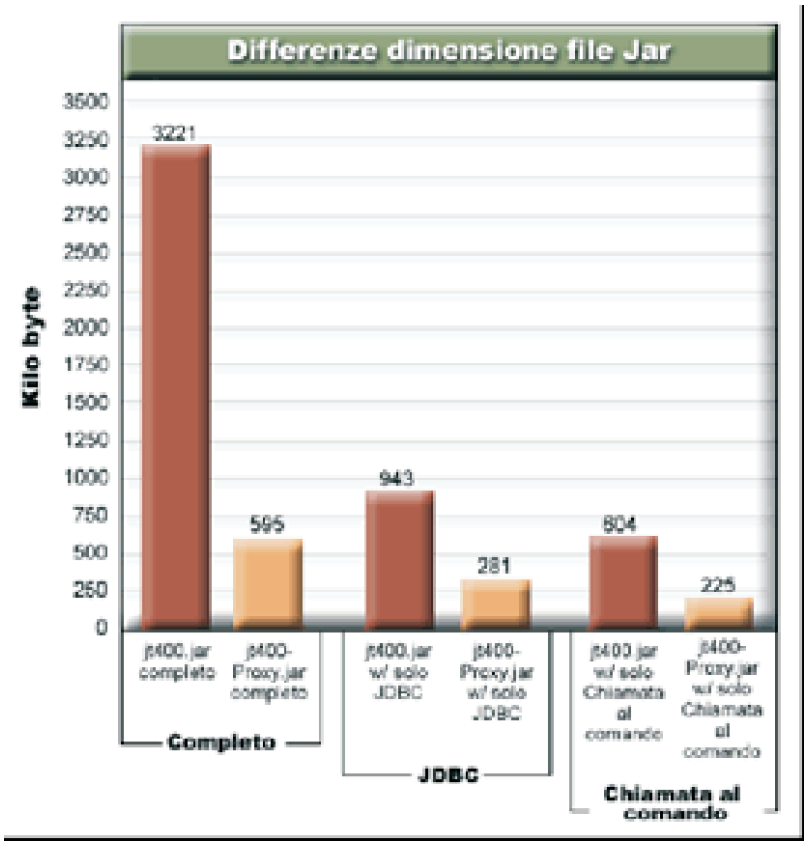
Prima del supporto proxy, le classi contenenti l'interfaccia pubblica, tutte le classi necessarie per l'elaborazione della richiesta e l'applicazione stessa erano eseguite sulla stessa JVM. Quando si utilizza il supporto proxy, è necessario che l'interfaccia pubblica sia associata all'applicazione, ma le classi per l'elaborazione delle richieste possano essere eseguite su una diversa JVM. Il supporto proxy non modifica l'interfaccia pubblica. E' possibile eseguire lo stesso programma con la versione proxy dell'IBM Toolbox per Java o con la versione standard.

Utilizzo del file jt400Proxy.jar

L'obiettivo dello scenario proxy a più livelli è quello di ridurre al massimo le dimensioni del file jar dell'interfaccia pubblica, cosicché sia possibile scaricarlo più velocemente da un'applet. Quando si utilizzano le classi proxy, non è necessario installare l'intero IBM Toolbox per Java sul client. Al contrario, utilizzare AS400JarMaker nel file jt400Proxy.jar per includere solo i componenti necessari per ridurre al massimo le dimensioni del file jar.

La Figura 2 confronta la dimensione dei file jar proxy con i file jar standard:

Figura 2: confronto della dimensione dei file jar proxy e dei file jar standard



Un ulteriore vantaggio è che il supporto proxy richiede di avere un minor numero di porte aperte in un firewall. Con la versione standard dell'IBM Toolbox per Java, è necessaria l'apertura di più porte. Questo accade perché, ogni servizio dell'IBM Toolbox per Java utilizza una porta diversa per comunicare con il server. Ad esempio, la chiamata al comando utilizza una porta diversa da JDBC, che utilizza invece una porta diversa dalla stampa e così via. E' necessario abilitare ognuna di queste porte nel firewall. Tuttavia, quando si utilizza il supporto proxy, tutti i dati passano attraverso la stessa porta.

Proxy standard e tunnel HTTP

Per l'esecuzione tramite proxy sono disponibili due opzioni: proxy standard e tunneling HTTP:

- Nel proxy standard il client proxy ed il server proxy comunicano utilizzando un socket su una porta. La porta predefinita è 3470. Modificare la porta predefinita utilizzando il metodo setPort() nella classe ProxyServer o utilizzando l'opzione `-port` durante l'avvio del server proxy.Ad esempio:

  ```
  java com.ibm.as400.access.ProxyServer -port 1234
  ```

- Nel tunnel HTTP il client proxy ed il server proxy comunicano tramite il server HTTP. L'IBM Toolbox per Java fornisce un servlet che gestisce la richiesta proxy. Il client proxy richiama il servlet tramite il

server HTTP. Il vantaggio del tunnel è che all'utente non viene richiesto di aprire un'altra porta nei firewall, poiché la comunicazione passa attraverso la porta HTTP. Lo svantaggio del tunnel consiste nella minore velocità rispetto al proxy standard.

L'IBM Toolbox per Java utilizza il nome del server proxy per stabilire se si sta utilizzando il proxy standard o il tunneling proxy:

- Per il proxy standard, utilizzare solo il nome del server. Ad esempio:

```
com.ibm.as400.access.AS400.proxyServer=myServer
```

- Per il tunneling, utilizzare un'URL per forzare il client proxy ad utilizzare il tunneling. Ad esempio:

```
com.ibm.as400.access.AS400.proxyServer=http://myServer
```

Durante l'esecuzione del proxy standard, esiste un collegamento socket fra il client ed il server. Se il collegamento ha esito negativo, il server elimina le risorse associate a quel client.

Quando si utilizza il tunnel HTTP, l'utilizzo del protocollo HTTP rende il proxy privo di collegamento. Cioè, viene stabilito un nuovo collegamento per ogni flusso di dati. Dal momento che il protocollo è privo di collegamento, il server non sa se l'applicazione client sia ancora attiva o meno. Di conseguenza, il server non sa quando eliminare le risorse. Il server di tunneling risolve tale problema utilizzando un sottoprocesso per eliminare le risorse ad intervalli prestabiliti (basati su un valore di supero tempo).

Alla fine dell'intervallo prestabilito, il sottoprocesso esegue ed elimina le risorse non utilizzate di recente. Due proprietà di sistema gestiscono il sottoprocesso:

- com.ibm.as400.access.TunnelProxyServer.clientCleanupInterval indica con quale frequenza, in secondi, venga eseguito il sottoprocesso di ripulitura.L'impostazione predefinita è ogni due ore.
- com.ibm.as400.access.TunnelProxyServer.clientLifetime indica per quanto tempo, in secondi, una risorsa può rimanere inattiva prima di venire eliminata.L'impostazione predefinita è 30 minuti.

Utilizzo del server proxy

Per utilizzare l'implementazione del server proxy relativa alle classi dell'IBM Toolbox per Java, completare le seguenti operazioni:

1. Eseguire AS400ToolboxJarMaker su jt400Proxy.jar per eliminare le classi non necessarie. Questa operazione è facoltativa ma consigliata.
2. Stabilire come richiamare jt400Proxy.jar nel client.
   - Per i programmi Java, utilizzare la classe AS400ToolboxInstaller o un altro metodo per richiamare il file nel client.
   - Per le applet Java, si potrebbe scaricare il file jar dal server HTML.
3. Determinare quale server verrà utilizzato per il server proxy.
   - Per le applicazioni Java, il server proxy può essere qualunque computer.
   - Per le applet Java, è necessario eseguire il server proxy sullo stesso computer del server HTTP.
4. Verificare di aver immesso jt400.jar nel CLASSPATH nel server.
5. Avviare il server proxy o utilizzare il servlet proxy:
   - Per il proxy standard, avviare il server proxy utilizzando il seguente comando:

     ```
     java com.ibm.as400.access.ProxyServer
     ```

   - Per il tunneling proxy, configurare il server HTTP per utilizzare il servlet proxy. Il nome della classe del servlet è com.ibm.as400.access.TunnelProxyServer ed è contenuta in jt400.jar.
6. Sul client, impostare una proprietà di sistema per identificare il server proxy.L'IBM Toolbox per Java si serve di tale proprietà di sistema per stabilire se si sta utilizzando il proxy standard o il tunneling proxy.
   - Per il proxy standard, il valore della proprietà è il nome della macchina che esegue il server proxy. Ad esempio:

```
com.ibm.as400.access.AS400.proxyServer=myServer
```

- Per il tunneling proxy, utilizzare un URL per forzare il client proxy ad utilizzare il tunnel. Ad esempio:

```
com.ibm.as400.access.AS400.proxyServer=http://myServer
```

7. Eseguire il programma client.

Quando si desidera gestire sia le classi proxy che le classi non contenute in jt400Proxy.jar, è possibile fare riferimento a jt400.jar invece che a jt400Proxy.jar. jt400Proxy.jar è una sottoserie di jt400.jar per cui tutte le classi proxy sono contenute nel file jt400.jar.

Utilizzo dell'SSL

Quando si utilizza il proxy, sono disponibili tre opzioni per la codifica dei dati nel passaggio dal client proxy al server iSeries di destinazione. Gli algoritmi SSL vengono utilizzati per codificare i dati.

1. E' possibile codificare i flussi di dati fra il client proxy ed il server proxy.
2. E' possibile codificare i flussi di dati fra il server proxy ed il server iSeries di destinazione.
3. Entrambi. E' possibile codificare il flusso di dati fra il client proxy ed il server proxy ed il flusso fra il server proxy ed il server iSeries di destinazione.

Per ulteriori informazioni, consultare Secure Sockets Layer.

Esempi: utilizzo dei server proxy

I seguenti rappresentano tre esempi specifici per l'utilizzo di un server proxy con i passi riportati sopra.

- Eseguire un'applicazione Java utilizzando il supporto proxy
- Eseguire un applet Java utilizzando il supporto proxy
- Eseguire un'applicazione Java utilizzando un supporto tunneling proxy.

Classi abilitate alla gestione di un server proxy

Alcune classi dell'IBM Toolbox per Java vengono abilitate alla gestione dell'applicazione del server proxy. Esse sono le seguenti:

- JDBC
- Accesso al livello del record
- Integrated file system
- Print
- Code di dati
- Chiamata al comando
- Chiamata al programma
- Chiamata al programma di servizio
- Spazio utente
- Area dati
- Classe AS400
- Classe SecureAS400

Al momento non vengono supportate altre classi dal jt400Proxy. Inoltre, le autorizzazioni dell'IFS non funzionano utilizzando solo il file jar proxy. Tuttavia, è possibile utilizzare la classe JarMaker per includere tali classi contenute nel file jt400.jar.

Descrizione lunga della Figura 1: come un client standard e un client proxy si collegano ad un server (rzahh505.gif)

In IBM Toolbox per Java: supporto Proxy

Questa figura illustra:
- come un client standard e un client proxy si collegano ad un server
- i file jar di IBM Toolbox per Java richiesti

Descrizione

La figura è composta da quanto segue:
- Un'immagine di un personal computer in alto a sinistra che rappresenta un client tradizionale (standard). Questa immagine include un cerchio che contiene il nome del file jar di IBM Toolbox per Java richiesto (jt400Proxy.jar).
- Un'immagine di un personal computer nella parte in basso a sinistra che rappresenta un client proxy. Questa immagine include un cerchio che contiene il nome del file jar di IBM Toolbox per Java richiesto.
- Un'immagine di un server iSeries in basso al centro che rappresenta un server proxy. Questa immagine include un cerchio che contiene il nome del file jar di IBM Toolbox per Java richiesto.
- Un'immagine di un server iSeries sulla destra che rappresenta il server iSeries. Il server iSeries non richiede un file jar, quindi non include un cerchio.
- Linee che collegano le immagini.

Al cliente tradizionale (standard) viene assegnata l'etichetta ″Client fat″ ed utilizza il file jt400.jar. Il client standard si collega direttamente al server iSeries.

Il client proxy utilizza il file jt400Proxy.jar. Il client proxy si collega al server proxy, che utilizza il file jt400.jar. Il server proxy si collega al server iSeries.

Descrizione lunga della Figura 1: confronto della dimensione dei file jar proxy rispetto ai file jar standard (rzahh502.gif)

In IBM Toolbox per Java: supporto Proxy

Questa figura è un grafico a barre che mette a confronto le dimensioni dei file jar standard e proxy, dopo aver utilizzato AS400JarMaker per ridurre le dimensioni dei file jar.

Descrizione

Il grafico effettua tre confronti tra il file jt400.jar ed il file jt400Proxy.jar:
- I file jar completi
- I file jar quando contengono solo il componente JDBC
- I file jar quando contengono solo il componente Chiamata al comando

Il grafico a barre è composto da un asse orizzontale (asse delle x) e da un asse verticale (asse delle y):
- L'asse orizzontale (o asse delle x) del grafico è composto da tre gruppi di barre che rappresentano i file jar.
- L'asse verticale (o delle asse y) del grafico rappresenta la dimensione dei file jar misurata in kilobyte.

I gruppi sono etichettati Completo, JDBC e Chiamata al comando. Ogni gruppo contiene una barra per il file jt400.jar e una barra per il file jt400Proxy.jar. Seguono le descrizioni per ogni gruppo:

- Completo: le dimensioni del file jar sono 3,221 kilobyte per il jt400.jar completo e 595 kilobyte per il jt400Proxy.jar completo.
- JDBC: le dimensioni del file jar sono 943 kilobyte per il jt400.jar e 281 kilobyte per il jt400Proxy.jar, ognuno include solo il componente JDBC.
- Chiamata al comando: le dimensioni del file jar sono 604 kilobyte per il jt400.jar e 225 kilobyte per il jt400Proxy.jar, ognuno include solo il componente Chiamata al comando.

Esempio: esecuzione di un'applicazione Java utilizzando il supporto Proxy

Il seguente esempio illustra le fasi dell'esecuzione di un'applicazione Java utilizzando il supporto proxy.

1. Selezionare una macchina che funzioni come server proxy. L'ambiente Java e CLASSPATH sulla macchina del server includono il file `jt400.jar`. Questa macchina deve essere in grado di collegarsi al server iSeries.
2. Avviare il server proxy su questa macchina immettendo: `java com.ibm.as400.access.ProxyServer -verbose`. La specifica di verbose consente di monitorare gli eventuali collegamenti e scollegamenti del client.
3. Selezionare una macchina che funzioni come client. L'ambiente Java e CLASSPATH sulla macchina client includono il file `jt400Proxy.jar` e le classi dell'applicazione. Questa macchina deve essere in grado di collegarsi al server proxy ma non è necessario che si colleghi al server iSeries.
4. Impostare il valore della proprietà di sistema `com.ibm.as400.access.AS400.proxyServer` in modo che sia il nome del server proxy ed eseguire l'applicazione. Un modo semplice di farlo è utilizzando l'opzione -D nella maggior parte dei richiami alla Java Virtual Machine: `java -Dcom.ibm.as400.access.AS400.proxyServer=psMachineName YourApplication`
5. Durante l'esecuzione dell'applicazione, si constata (se si imposta verbose nel passo 2) che l'applicazione effettua almeno un collegamento al server proxy.

Esempio: esecuzione di un'applet Java utilizzando un supporto proxy

Il seguente esempio illustra le fasi dell'esecuzione di un'applet Java utilizzando il supporto proxy.

1. Selezionare una macchina che funzioni come server proxy. Le applet possono inizializzare i collegamenti di rete solo sulla macchina da cui sono state scaricate in origine; quindi, è preferibile eseguire il server proxy sulla stessa macchina del server HTTP. L'ambiente Java e CLASSPATH nella macchina del server proxy includono il file jt400.jar.
2. Avviare il server proxy su questa macchina immettendo: `java com.ibm.as400.access.ProxyServer -verbose`. La specifica di verbose consentirà di monitorare gli eventuali collegamenti e scollegamenti del client.
3. E' necessario scaricare il codice Applet prima che venga eseguito per questa ragione è meglio ridurre al massimo la dimensione del codice. L'AS400ToolboxJarMaker può ridurre jt400Proxy.jar in modo significativo includendo solo il codice per i componenti utilizzati dall'applet. Ad esempio, se un'applet utilizza solo JDBC, ridurre il file jt400Proxy.jar in modo da includere la minima quantità di codice eseguendo questo comando:

   ```
   java utilities.AS400ToolboxJarMaker -source jt400Proxy.jar -destination jt400ProxySmall.jar
                                       -component JDBC
   ```

4. L'applet deve impostare il valore della proprietà di sistema com.ibm.as400.access.AS400.proxyServer in modo che sia il nome del server proxy. Un modo appropriato in cui le applet possono farlo è utilizzare una classe Properties compilata (Esempio). Compilare questa classe e ubicare il file Properties.class creato nell'indirizzario com/ibm/as400/access (lo stesso percorso da cui proviene il file html). Ad esempio, se il file html è /mystuff/HelloWorld.html, Properties.class si trova in /mystuff/com/ibm/as400/access.
5. Inserire jt400ProxySmall.jar nello stesso indirizzario del file html (/mystuff/ nel passo 4).
6. Fare riferimento all'applet nel file HTML:

```
<APPLET archive="jt400Proxy.jar, Properties.class" code="YourApplet.class"
        width=300 height=100> </APPLET>
```

Esempio: esecuzione di un'applicazione Java utilizzando il supporto Tunneling Proxy

Il seguente esempio illustra le fasi dell'esecuzione di un'applicazione Java utilizzando il supporto tunneling proxy.

1. Scegliere il server HTTP sul quale si desidera eseguire il server proxy, quindi configurarlo per eseguire il servlet com.ibm.as400.access.TunnelProxyServer (in jt400.jar).**Nota:** assicurarsi che il server HTTP abbia un collegamento al server iSeries che contiene i dati o la risorsa che l'applicazione utilizza dal momento che il servlet si collega a tale iSeries per effettuare le richieste.
2. Scegliere una macchina che funzioni come client e verificare che il CLASSPATH sulla macchina client includa il file jt400Proxy.jar e le classi dell'applicazione. Il client deve essere in grado di collegarsi al server HTTP ma non è necessario che si colleghi al server iSeries.
3. Impostare il valore della proprietà com.ibm.as400.access.AS400.proxyServer in modo che sia il nome del server HTTP nel formato URL.
4. Eseguire l'applicazione, impostando il valore della proprietà com.ibm.as400.access.AS400.proxyServer in modo che sia il nome del server HTTP in formato URL. E' possibile effettuare facilmente tale operazione utilizzando l'opzione -D presente nella maggior parte delle JVM:

   ```
   java -Dcom.ibm.as400.access.AS400.proxyServer=http://psMachineName YourApplication
   ```

   Nota: il codice del client proxy crea l'URL del servlet appropriata concatenando "servlet" ed il nome del servlet al nome del server. In questo esempio, esso converte http://psMachineName in http://psMachineName/servlet/TunnelProxyServer

---

SSL (Secure Sockets Layer) e JSSE (Java Secure Socket Extension)

IBM Toolbox per Java supporta l'utilizzo di JSSE (Java Secure Socket Extension) per i collegamenti SSL Java (Secure Sockets Layer). JSSE è disponibile come pacchetto facoltativo per Java 2 Platform, Standard Edition (J2SE), versioni 1.2 e 1.3. JSSE viene integrato in J2SE, versione 1.4.

Per ulteriori informazioni relative a JSSE, consultare il sito web Sun JSSE  .

JSSE fornisce la capacità di eseguire l'autenticazione del server, consentire comunicazioni sicure e codificare dati. Utilizzando JSSE, è possibile fornire scambi di dati sicuri tra i client e i server in cui è in esecuzione un qualsiasi protocollo di applicazione (ad esempio, HTTP e FTP) su TCP/IP.

Dopo aver installato e configurato JSSE, IBM Toolbox per Java lo utilizza per impostazione predefinita. Sarebbe necessario eseguire la migrazione a JSSE dal momento che sslight non sarà più aggiornato. Informazioni relative all'utilizzo di sslight per abilitare SSL sono incluse solo per la compatibilità con le versioni precedenti.

Prima di iniziare ad utilizzare SSL con IBM Toolbox per Java, è necessario considerare le responsabilità legali.

IBM Toolbox per Java 2 Micro Edition

Il pacchetto IBM Toolbox per Java 2 Micro Edition (com.ibm.as400.micro) consente di scrivere programmi Java che permettono a varie unità senza fili Tier0, come PDA (personal digital assistant) e cellulari, di accedere direttamente ai dati e alle risorse iSeries.

Per ulteriori informazioni su ToolboxME per iSeries, consultare i seguenti argomenti:

Requisiti
Acquisire informazioni per sviluppare applicazioni con ToolboxME per iSeries e per eseguire quelle applicazioni sulle unità Tier0.

Scaricamento ed impostazione di ToolboxME per iSeries
Acquisire informazioni su come scaricare ed installare ToolboxME per iSeries sul server, sulla stazione di lavoro e sull'unità Tier0.

Concetti
Leggere una breve introduzione che definisce concetti importanti per lo sviluppo di applicazioni che si eseguono sulle unità Tier0.

Classi ToolboxME per iSeries
Leggere informazioni circa le classi nel componente ToolboxME per iSeries (pacchetto com.ibm.as400.micro). Le classi forniscono una serie di funzioni ridotta disponibile nelle classi Access di IBM Toolbox per Java, nel supporto JDBC e altro.

Creazione di un programma ToolboxME per iSeries
Acquisire nozioni sulle operazioni per la creazione di programmi ToolboxME per iSeries che si eseguono sull'unità Tier0. Seguire le istruzioni e creare il primo programma ToolboxME per iSeries.

Esempi
Esaminare, scaricare ed eseguire gli esempi di lavoro di ToolboxMe per iSeries utili per comprendere come creare ed utilizzare applicazioni senza fili.

Scaricamento ed impostazione di ToolboxME per iSeries

E' necessario scaricare separatamente ToolboxME per iSeries (jt400Micro.jar), che è contenuto in JTOpen.

E' possibile scaricare ToolboxME per iSeries dal sito web IBM Toolbox per Java/JTOpen  che offre anche informazioni aggiuntive sull'installazione di ToolboxME per iSeries.

L'installazione di ToolboxME per iSeries è differente per l'unità Tier0, la stazione di lavoro di sviluppo ed il server:

- Creare una applicazione per l'unità senza fili (utilizzando jt400Micro.jar) ed istallare l'applicazione come documentato dal produttore dell'unità.
- Assicurarsi che i Server host di iSeries siano avviati sul server che contiene i dati di destinazione.
- Assicurarsi che il sistema sul quale si desidera eseguire MEServer abbia accesso a jt400.jar.

Per ulteriori informazioni, consultare le seguenti pagine:

"Installazione di IBM Toolbox per Java sulla stazione di lavoro" a pagina 13

"Installazione di IBM Toolbox per Java su un server iSeries" a pagina 12

Concetti importanti per l'utilizzo di ToolboxME per iSeries

Prima di iniziare lo sviluppo delle applicazioni Java ToolboxME per iSeries, è necessario comprendere i seguenti concetti e standard che guidano tale sviluppo.

J2ME (Java 2 Platform, Micro Edition)

J2ME(TM) è l'implementazione dello standard Java 2 che fornisce gli ambienti tempo di esecuzione di Java per le unità senza fili Tier0, come i PDA (personal digital assistant) e i cellulari.IBM Toolbox per Java 2 Micro Edition aderisce a questo standard.

Unità Tier0

Le unità senza fili, come ad esempio PDA e cellulari, che utilizzano la tecnologia senza fili per collegarsi a computer e reti, vengono considerate unità Tier0. Questo nome si basa sul modello comune di applicazione a 3 livelli. Il modello a 3 livelli descrive un programma distribuito che è organizzato in tre parti principali, ognuna delle quali si trova su un diverso computer o rete:

- Il terzo livello rappresenta il database e i programmi relativi che si trovano su un server, spesso differente rispetto al secondo livello. Questo livello fornisce le informazioni e l'accesso a quelle informazioni che gli altri livelli utilizzano per eseguire un lavoro.
- Il secondo livello rappresenta la logica aziendale, che solitamente si trova su un computer differente, solitamente un server, condiviso su una rete.
- Il primo livello è generalmente la parte dell'applicazione che si trova su una stazione di lavoro, inclusa l'interfaccia utente.

Spesso le unità Tier0 sono piccole, portatili, con risorse limitate, come i PDA ed i cellulari. Le unità Tier0 sostituiscono o integrano la funzionalità delle unità al primo livello.

CLDC (Connected Limited Device Configuration)

Una configurazione definisce una serie minima di API e le capacità necessarie di una JVM per fornire le funzioni previste per un'ampia serie di unità. Il CLDC si rivolge ad una ampia serie di unità con risorse limitate che comprendono le unità Tier0.

Per ulteriori informazioni, consultare CLDC  .

MIDP (Mobile Information Device Profile)

Un profilo rappresenta una serie di API create in base ad una configurazione esistente che si rivolgono ad un determinato tipo di unità o sistema operativo. Il MIDP, creato in base alla CLDC, fornisce un ambiente tempo di esecuzione standard che consente di distribuire dinamicamente le applicazioni ed i servizi alle unità Tier0.

Per ulteriori informazioni, consultare MIDP (Mobile Information Device Profile)  .

JVM per unità senza fili

Per eseguire l'applicazione Java, l'unità Tier0 richiede una JVM progettata appositamente per le risorse limitate di una unità senza fili. Alcune delle possibili JVM che è possibile utilizzare includono le seguenti:

- IBM J9 virtual machine, parte dell'ambiente IBM WebSphere Micro 
- Sun K Virtual Machine (KVM), parte di CLDC 
- MIDP 

Informazioni correlate

E' possibile utilizzare uno qualsiasi dei numerosi strumenti di sviluppo creati per supportare la creazione di applicazioni Java senza fili. Per un breve elenco di tali strumenti, consultare Informazioni correlate a IBM Toolbox per Java.

Per saperne di più e per scaricare simulazioni ed emulazioni di unità senza fili, consultare il sito web per l'unità o per il sistema operativo sul quale si desidera eseguire l'applicazione.

Classi ToolboxME per iSeries

Il com.ibm.as400.micro package fornisce le classi necessarie per scrivere le applicazioni che consentono a "Unità Tier0" a pagina 354 di accedere ai dati e alle risorse del server iSeries.

Nota: per utilizzare le classi ToolboxMe per iSeries, è necessario scaricare ed impostare separatamente il componente ToolboxME per iSeries.

ToolboxME per iSeries fornisce le seguenti classi:

- "Classe MEServer" indirizza le richieste dall'unità Tier0 al server host
- Numerose classi forniscono una sottoserie di funzioni dal pacchetto Access di IBM Toolbox per Java
  - "Classe AS400" a pagina 356 si collega a un server iSeries
  - "Classe CommandCall" a pagina 356 richiama un comando iSeries
  - "Classe DataQueue" a pagina 357 legge e scrive i dati su una coda dati del server iSeries
  - "Classe ProgramCall" a pagina 358 richiama un programma del server iSeries e accede ai dati restituiti dopo l'esecuzione del programma
- "Classi JdbcMe" a pagina 359 fornisce il supporto JDBC includendo la più piccola serie utile di metodi e dati dal pacchetto java.sql

Classe MEServer

Utilizzare la classe MEServer per adempiere alle richieste dalla propria applicazione client Tier0 che utilizza i file jar di ToolboxME per iSeries. MEServer crea gli oggetti IBM Toolbox per Java e richiama i metodi su tali oggetti per conto dell'applicazione client.

Nota: per utilizzare le classi ToolboxMe per iSeries, è necessario scaricare e impostare separatamente il componente ToolboxME per iSeries. Per ulteriori informazioni, consultare Scaricare ed installare ToolboxME per iSeries.

Utilizzare il seguente comando per avviare un MEServer:

```
java com.ibm.as400.micro.MEServer [options]
```

dove [options] rappresenta una o più delle seguenti opzioni:

-pcml pcml_doc1 [;pcml_doc2;...]
Specifica il documento PCML da precaricare ed analizzare. E' possibile abbreviare questa opzione utilizzando `-pc`.

Per importanti informazioni sull'utilizzo di questa opzione, consultare il Javadoc di MEServer .

-port port
Specifica la porta da utilizzare per accettare i collegamenti dai client. La porta predefinita è 3470. E' possibile abbreviare questa opzione utilizzando `-po`.

-verbose [true|false]
Specifica se stampare le informazioni sullo stato e sul collegamento in System.out. E' possibile abbreviare questa opzione utilizzando `-v`.

-help Stampa informazioni sull'utilizzo in System.out. E' possibile abbreviare questa opzione utilizzando `-h` o `-?`.Non è necessario il valore predefinito per stampare le informazioni sull'utilizzo.

MEServer non si avvierà se un altro server è attivo sulla porta specificata.

Classe AS400

La classe AS400 in un pacchetto micro (com.ibm.as400.micro.AS400) fornisce una sottoserie modificata delle funzioni disponibili nella classe AS400 nel pacchetto Access (com.ibm.as400.access.AS400). Utilizzare la classe AS400 di ToolboxMe per iSeries per collegarsi a un server iSeries da una unità Tier0.

Nota: per utilizzare le classi ToolboxMe per iSeries, è necessario scaricare e impostare separatamente il componente ToolboxME per iSeries. Per ulteriori informazioni, consultare Scaricare ed installare ToolboxME per iSeries.

La classe AS400 fornisce le seguenti funzioni:

- Collegarsi a MEServer
- Scollegarsi da MEServer

Il collegamento a MEServer viene effettuato implicitamente. Ad esempio, dopo aver creato un oggetto AS400, è possibile utilizzare il metodo run() in CommandCall per eseguire automaticamente connect(). In altre parole, non è necessario richiamare esplicitamente il metodo connect() a meno che non si voglia controllare quando viene stabilito il collegamento.

Esempio: utilizzo della classe AS400

Il seguente esempio mostra come utilizzare la classe AS400 per collegarsi a un server iSeries:

```
    AS400 system = new AS400("mySystem", "myUserid", "myPwd", "myMEServer");
                            try
{
          system.connect();
}
                            catch (Exception e)
{
           // Gestire l'eccezione
}
    // Eseguito con l'oggetto di sistema.
  system.disconnect();
```

Classe CommandCall

La classe CommandCall nel pacchetto micro (com.ibm.as400.micro.CommandCall) fornisce una sottoserie modificata delle funzioni disponibili nella classe CommandCall nel pacchetto Access (com.ibm.as400.access.CommandCall).Utilizzare la classe CommandCall per effettuare la chiamata a un comando iSeries da una unità Tier0 .

Nota: per utilizzare le classi ToolboxMe per iSeries, è necessario scaricare ed impostare separatamente il componente ToolboxME per iSeries.

Il metodo CommandCall run() richiede una Stringa (il comando che si vuole eseguire) e restituisce tutti i messaggi che risultano dall'esecuzione del comando come Stringa.Se il comando si completa ma non crea messaggi, il metodo run() restituisce una schiera Stringa vuota.

Esempio: utilizzo di CommandCall

Il seguente esempio dimostra come si può utilizzare CommandCall:

```
   // Gestire i comandi.
   AS400 system = new AS400("mySystem", "myUserid", "myPwd", "myMEServer");
                            try
{
    // Eseguire il comando "CRTLIB FRED."
    String[] messages = CommandCall.run(system, "CRTLIB FRED");
           if (messages != null)
    {
                   // Notare che si è verificato un errore.
        System.out.println("Command failed:");
                   for (int i = 0; i < messages.length; ++i)
        {
                           System.out.println(messages[i]);
        }
    }
                            else
    {
        System.out.println("Command succeeded!");
    }
}
                            catch (Exception e)
{
          // Gestire l'eccezione
}
    // Eseguito con l'oggetto di sistema.
  system.disconnect();
```

Classe DataQueue

La classe DataQueue nel pacchetto micro (com.ibm.as400.micro.DataQueue) fornisce una sottoserie modificata di funzioni disponibili nella classe DataQueue nel pacchetto Access (com.ibm.as400.access.DataQueue). Utilizzare la classe DataQueue per fare in modo che l'unità Tier0 o scrivere legga da o scriva su una coda dati nel server iSeries.

Nota: per utilizzare le classi ToolboxMe per iSeries, è necessario scaricare ed impostare separatamente il componente ToolboxME per iSeries.

La classe DataQueue include i seguenti metodi:

- Leggere o scrivere una voce come Stringa
- Leggere o scrivere una voce come una schiera di byte

Per leggere o scrivere voci, è necessario fornire il nome del server iSeries su cui si trova la coda dati ed il nome percorso IFS completo della coda dati. Quando non sono disponibili voci, la lettura di una voce restituisce un valore nullo.

Esempio: utilizzo di DataQueue per leggere e scrivere i dati su una coda

Il seguente esempio mostra come utilizzare la classe DataQueue per leggere e scrivere le voci su una coda dati su un server iSeries:

```
    AS400 system = new AS400("mySystem", "myUserid", "myPwd", "myMEServer");
                            try
{
          // Scrivere nella coda dati.
    DataQueue.write(system, "/QSYS.LIB/FRED.LIB/MYDTAQ.DTAQ", "some text");

          // Leggere dalla coda dati.
    String txt = DataQueue.read(system, "/QSYS.LIB/FRED.LIB/MYDTAQ.DTAQ");
}
                            catch (Exception e)
{
```

```
            // Gestire l'eccezione
}
    // Eseguito con l'oggetto di sistema.
   system.disconnect();
```

Classe ProgramCall

La classe ProgramCall nel pacchetto micro (com.ibm.as400.micro.ProgramCall) fornisce una sottoserie modificata delle funzioni disponibili nella classe ProgramCall nel pacchetto Access e (com.ibm.as400.access.ProgramCall). Utilizzare la classe ProgramCall per consentire ad una unità Tier0 di richiamare un programma iSeries ed accedere ai dati che vengono restituiti dopo che il programma viene eseguito.

Nota: per utilizzare le classi ToolboxMe per iSeries, è necessario scaricare e impostare separatamente il componente ToolboxME per iSeries. Per ulteriori informazioni, consultare "Requisiti ToolboxME per iSeries" a pagina 10.

Per utilizzare il metodo ProgramCall.run(), è necessario fornire i seguenti parametri:

- Il server su cui si desidera eseguire il programma
- Il nome del documento "PMCL (Program Call Markup Language)" a pagina 381
- Il nome del programma che si desidera eseguire
- La tabella hash che contiene il nome di uno o più parametri di programma che si desidera impostare ed i valori associati
- La schiera di stringa che contiene il nome di tutti i parametri da restituire dopo l'esecuzione del programma

ProgramCall utilizza PCML per descrivere i parametri di immissione ed emissione del programma. Il file PCML deve essere sulla stessa macchina dell'MEServer ed è necessario avere una voce per l'indirizzario che contiene il file PCML nel CLASSPATH di quella macchina.

L'utente deve registrare ogni documento PCML con l'MEServer. La registrazione di un documento PCML significa comunicare all'MEServer il programma definito da PCML che si desidera eseguire. Registrare il documento PCML durante il tempo di esecuzione o quando si avvia l'MEServer.

Per ulteriori informazioni sulla tabella hash che contiene i parametri del programma o su come registrare un documento PCML, consultare Javadoc ProgramCall di ToolboxME per iSeries. Per ulteriori informazioni su PCML, consultare "PMCL (Program Call Markup Language)" a pagina 381.

Esempio: utilizzo di ProgramCall

Il seguente esempio mostra come usare la classe ProgramCall per utilizzare l'unità Tier0 per eseguire un programma su un server:

```
// Chiamare i programmi.
AS400 system = new AS400("mySystem", "myUserid", "myPwd", "myMEServer");

String pcmlName = "qsyrusri.pcml"; // Il documento PCML che descrive il programma da utilizzare.
String apiName = "qsyrusri";

Hashtable parametersToSet = new Hashtable();
parametersToSet.put("qsyrusri.receiverLength", "2048");
parametersToSet.put("qsyrusri.profileName", "JOHNDOE" };

String[] parametersToGet = { "qsyrusri.receiver.userProfile",
                           "qsyrusri.receiver.previousSignonDate",
                           "qsyrusri.receiver.previousSignonTime",
                                        "qsyrusri.receiver.displaySignonInfo" };

String[] valuesToGet = null;
```

```
                            try
{
           valuesToGet = ProgramCall.run(system, pcmlName, apiName, parametersToSet, parametersToGet);

           // Richiamare e visualizzare il profilo utente.
           System.out.println("User profile: " + valuesToGet[0]);

           // Richiamare e visualizzare la data in un formato leggibile.
    char[] c = valuesToGet[1].toCharArray();
           System.out.println("Last Signon Date: " + c[3]+c[4]+"/"+c[5]+c[6]+"/"+c[1]+c[2] );

           // Richiamare e visualizzare l'ora in un formato leggibile.
    char[] d = valuesToGet[2].toCharArray();
           System.out.println("Last Signon Time: " + d[0]+d[1]+":"+d[2]+d[3]);

           // Richiamare e visualizzare le informazioni di collegamento.
           System.out.println("Signon Info: " + valuesToGet[3] );
}
   catch (MEException te)
{
           // Gestire l'eccezione.
}
   catch (IOException ioe)
{
           // Gestire l'eccezione
}

    // Eseguito con l'oggetto di sistema.
   system.disconnect();
```

Classi JdbcMe

Le classi ToolboxME per iSeries forniscono il supporto JDBC, incluso il supporto per il pacchetto java.sql. Le classi sono progettate per l'utilizzo in un programma che viene eseguito su un'unità Tier 0.

Le seguenti sessioni trattano l'accesso e l'utilizzo dei dati e descrivono il contenuto di JdbcMe, inclusi i collegamenti alle informazioni sulle singole classi JdbcMe.

Accesso ed utilizzo dei dati

Quando si utilizza un'unità Tier0 per accedere ed aggiornare i dati, è necessario che funzioni esattamente come se si stesse seduti davanti al sistema nel proprio ufficio. Tuttavia, molti degli sviluppi nelle unità Tier0 si concentrano sulla sincronizzazione dei dati. Utilizzando la sincronizzazione dei dati, ogni unità Tier0 possiede una copia di dati specifici provenienti dal database principale. Periodicamente, gli utenti sincronizzano i dati su ogni unità con il database principale.

La sincronizzazione dei dati non funziona al meglio con i dati dinamici. La gestione dei dati dinamici richiede l'accesso rapido ai dati aggiornati. L'attesa per accedere ai dati sincronizzati non è una opzione possibile per molte aziende. Inoltre, le esigenze software e hardware per i server e le unità per i principali dati sincronizzati possono essere rilevanti.

Per risolvere i problemi riguardanti il modello di sincronizzazione dei dati, le classi JdbcMe in ToolboxME per iSeries consentono di eseguire aggiornamenti in tempo reale ed accedere al database principale, ma permette ancora la memorizzazione di dati scollegati. L'applicazione può avere accesso ai dati importanti scollegati senza rendere gli aggiornamenti in tempo reale immediatamente parte del database principale. Questo tipo di approccio intermedio fornisce i vantaggi sia del modello dati sincronizzati che del modello dati in tempo reale.

Contenuto di JdbcMe

Per definizione, un programma di controllo di qualsiasi tipo per una unità Tier0 deve avere dimensioni molto ridotte. L'API di JDBC, tuttavia, è di dimensioni considerevoli. Le classi JdbcMe devono essere estremamente piccole ma comunque in grado di supportare un numero sufficiente di interfacce JDBC necessarie alle unità Tier0 per eseguire un lavoro significativo.

Le classi JdbcMe mettono a disposizione la funzionalità JDBC:
- La capacità di immettere o aggiornare dati
- Il controllo transazioni e la capacità di modificare i livelli di isolamento delle transazioni
- Serie di risultati che si possono scorrere e aggiornare
- Supporto SQL per le chiamate alle procedure memorizzate ed i trigger del programma di controllo

In più, le classi JdbcMe includono alcune caratteristiche uniche:
- Un programma di controllo universale che consente alla maggior parte dei dettagli di configurazione di essere consolidati in un singolo punto sul lato server
- Un meccanismo standard per conservare i dati sulla memoria scollegata

JdbcMe include le seguenti classi:
- JdbcMeConnection
- JdbcMeDriver
- JdbcMeException
- JdbcMeLiveResultSet
- JdbcMeOfflineData
- JdbcMeOfflineResultSet
- JdbcMeResultSetMetaData
- JdbcMeStatement

ToolboxME per iSeries fornisce un pacchetto java.sql che segue la specifica JDBC ma contiene solo la più piccola serie di classi e metodi utili. La fornitura di una serie minima di funzioni sql consente alle classi JdbcMe di essere piccole nelle dimensioni ma comunque sufficienti per eseguire le comuni operazioni JDBC.

Utilizzo di ToolboxME per iSeries per collegarsi al database sul server host:

La classe JdbcMeConnection fornisce una sottoserie di funzioni disponibili nella classe AS400JDBCConnection di IBM Toolbox per Java. Utilizzare JdbcMeConnection per consentire all'unità Tier0 di accedere ai database UDB (Universal Database) DB2 sul server host.

Nota: per utilizzare le classi ToolboxMe per iSeries, è necessario scaricare e impostare separatamente il componente ToolboxME per iSeries. Per ulteriori informazioni, consultare Requisiti ed istallazione di ToolboxME per iSeries.

Utilizzare JdbcMeDriver.getConnection() per collegarsi al database del server.Il metodo getConnection() richiede una stringa URL come argomento, l'ID utente e la parola d'ordine. Il gestore dell'unità di controllo JDBC sul server host tenta di localizzare un programma di controllo che si possa collegare al database rappresentato dalla URL. JdbcMeDriver utilizza la seguente sintassi per l'URL:

```
jdbc:as400://server-name/default-schema;meserver=<server>[:port];[other properties];
```

Nota: il precedente esempio di sintassi si trova su due righe, quindi è possibile visualizzarlo e stamparlo facilmente. Solitamente, l'URL viene visualizzata su una riga senza interruzioni e spazi aggiuntivi.

E' necessario specificare un nome server o JdbcMeDriver invia un'eccezione. Lo schema predefinito è facoltativo. Se non viene specificata una porta, JdbcMeDriver utilizza la porta 3470. Inoltre, è possibile impostare varie proprietà JDBC sulla URL. Per impostare le proprietà, utilizzare la seguente sintassi:

```
name1=value1;name2=value2;...
```

Consultare Proprietà JDBC per un elenco completo delle proprietà supportate da JdbcMeDriver.

Esempi: utilizzo di JdbcMeDriver per collegarsi al server

Esempio: collegamento al database del server senza specificare uno schema predefinito, una porta o le proprietà JDBC

Negli esempi viene specificato un ID utente e una parola d'ordine come parametri nel metodo:

```
       // Effettuare il collegamento al sistema 'mysystem'. Nessuno schema predefinito, porta o
               // proprietà specificate.
  Connection c  = JdbcMeDriver.getConnection("jdbc:as400://mysystem.helloworld.com;meserver=myMeServer;"
                                                    "auser",
                                                                "apassword");
```

Esempio: collegamento al database del server quando si specifica lo schema e le proprietà JDBC

Nell'esempio viene specificato un ID utente e una parola d'ordine come parametri sul metodo:

```
       // Effettuare il collegamento al sistema 'mysystem'. Specificare uno schema e
       // e due proprietà JDBC. Non specificare una porta.
  Connection c2 = JdbcMeDriver.getConnection(
     "jdbc:as400://mysystem.helloworld.com/mySchema;meserver=myMeServer;naming=system;errors=full;"
               "auser",
                           "apassword");
```

Esempio: collegamento al database del server

Nell'esempio, vengono specificate le proprietà (incluso l'ID utente e la parola d'ordine) tramite l'utilizzo di un'URL:

```
      // Stabilire un collegamento utilizzando le proprietà. Le proprietà vengono impostate sull'URL
      // invece che tramite l'oggetto proprietà.
  Connection c = DriverManager.getConnection(
     "jdbc:as400://mySystem;meserver=myMeServer;naming=sql;errors=full;user=auser;password=apassword");
```

Esempio: scollegamento dal database

Nell'esempio viene utilizzato il metodo close() sull'oggetto di collegamento per lo scollegamento dal server:

```
  c.close();
```

Classe JdbcMeDriver:

La classe JdbcMeDriver fornisce una sottoserie di funzioni disponibili nella classe AS400JDBCDriver di IBM Toolbox per Java. Utilizzare JdbcMeDriver nell'applicazione client di Tier0 per eseguire semplici istruzioni SQL che non hanno parametri ed ottenere ResultSets prodotti dalle istruzioni.

Nota: per utilizzare le classi ToolboxMe per iSeries, è necessario scaricare e impostare separatamente il componente ToolboxME per iSeries. Per ulteriori informazioni, consultare "Scaricamento ed impostazione di ToolboxME per iSeries" a pagina 353.

L'utente non registra esplicitamente il JdbcMeDriver; invece la proprietà **programma di controllo** specificata sulla URL nel metodo JdbcMeConnection.getConnection() determina il programma di

controllo. Ad esempio, per caricare l'unità di controllo JDBC di IBM Developer Kit per Java (denominata unità di controllo 'nativa'), utilizzare un codice simile al seguente:

```
Connection c = JdbcMeDriver.getConnection(
   "jdbc:as400://mysystem.myworld.com;meserver=myMeSrvr;driver=native;user=auser;password=apassword");
```

L'unità di controllo JDBC di IBM Toolbox per Java non richiede un oggetto AS400 come parametro di immissione come accade per le altre classi IBM Toolbox per Java che richiamano dati da un server. Tuttavia, un oggetto AS400 viene utilizzato internamente ed è necessario fornire esplicitamente un ID utente e una parola d'ordine. Fornire l'ID utente e la parola d'ordine nell'URL o tramite i parametri del metodo getConnection().

Per esempi sull'utilizzo di getConnection(), consultare JDBCMeConnection.

ResultSet:

Le classi ResultSet di ToolboxME per iSeries sono:

- JdbcMeLiveResultSet
- JdbcMeOfflineResultSet
- JdbcMeResultSetMetaData

JdbcMeLiveResultSet e JdbcMeOfflineResultSet contengono la stessa funzionalità, a parte il fatto che:

- JdbcMeLiveResultSet richiama i dati effettuando una chiamata al database sul server
- JdbcMeOfflineResultSet richiama i dati dal database sull'unità locale

Nota: per utilizzare le classi ToolboxMe per iSeries, è necessario scaricare e impostare separatamente il componente ToolboxME per iSeries. Per ulteriori informazioni, consultare Scaricare ed installare ToolboxME per iSeries.

JdbcMeLiveResultSet

La classe JdbcMeLiveResultSet fornisce una sottoserie di funzioni disponibili nella classe AS400JDBCResultSet di IBM Toolbox per Java. Utilizzare JdbcMeLiveResultSet nell'applicazione client Tier0 per accedere ad una tabella di dati creata tramite l'esecuzione di una interrogazione.

JdbcMeLiveResultSet richiama le righe della tabella in sequenza. All'interno di una riga, è possibile accedere ai valori della colonna in un qualsiasi ordine. JdbcMeLiveResultSet include i metodi che consentono di svolgere le seguenti operazioni:

- Richiamare i dati di vari tipi memorizzati nella serie di risultati
- Spostare il cursore alla riga specificata (riga precedente, riga corrente, riga successiva e così via)
- Inserire, aggiornare e cancellare righe
- Aggiornare colonne (utilizzando valori string e int)
- Richiamare l'oggetto ResultSetMetaData che descrive le colonne nella serie di risultati

Un cursore, che è un puntatore interno, viene utilizzato da una serie risultati per indicare la riga nella serie risultati alla quale accede il programma Java.JDBC 2.0 fornisce metodi aggiuntivi per accedere a specifiche posizioni in un database:

Posizioni del cursore scorrevole
absolute first last moveToCurrentRow moveToInsertRow previous relative

Capacità di scorrimento

Se una serie di risultati viene creata eseguendo un'istruzione, è possibile spostarsi (scorrere) all'indietro (dall'ultimo al primo) o in avanti (dal primo all'ultimo) attraverso le righe in una tabella.

Una serie risultati che supporta questo spostamento viene denominata serie risultati scorrevole. Le serie di risultati scorrevoli supportano anche il posizionamento relativo ed assoluto. Il posizionamento relativo consente di spostarsi ad una riga nella serie di risultati specificando una posizione relativa alla riga corrente. Il posizionamento assoluto consente di spostarsi direttamente alla riga specificando la sua posizione nella serie di risultati.

Con JDBC 2.0, si hanno a disposizione due ulteriori capacità di scorrimento da utilizzare quando si gestisce la classe ResultSet: serie di risultati insensibile allo scorrimento e sensibile allo scorrimento.

Una serie di risultati non sensibile allo scorrimento solitamente non è sensibile alle modifiche effettuate quando è aperta, mentre la serie di risultati sensibile allo scorrimento è sensibile alle modifiche. L'unità di controllo JDBC di IBM Toolbox per Java non supporta serie di risultati non sensibili allo scorrimento.

Serie di risultati aggiornabili

Nell'applicazione, è possibile utilizzare le serie di risultati che usano la combinazione di sola lettura (non è possibile aggiornare i dati) o la combinazione aggiornabile (consente aggiornamenti ai dati e può utilizzare blocchi di scrittura del database per controllare l'accesso alle stesse voci di dati da parte di differenti transazioni). In una serie di risultati aggiornabile, le righe possono essere aggiornate, inserite e cancellate.

Consultare Riepilogo metodi per un elenco completo di metodi di aggiornamento disponibili in JdbcMeResultSet.

Esempio: serie di risultati aggiornabili

Il seguente esempio mostra come utilizzare una serie di risultati che consente aggiornamenti ai dati (combinazione aggiornamento) e modifiche alla serie di risultati quando la serie dei risultati è aperta (sensibile allo scorrimento).

```
                    // Collegarsi al server.
     Connection c = JdbcMeDriver.getConnection(
        "jdbc:as400://mySystem;meserver=myMeServer;user=auser;password=apassword");

                    // Creare un oggetto Statement.
Impostare la combinazione della serie di risultati su
                    // aggiornabile.
     Statement s = c.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE, ResultSet.CONCUR_UPDATABLE);

          // Eseguire un'interrogazione. Il risultato viene inserito
                    // in un oggetto ResultSet.
     ResultSet rs = s.executeQuery ("SELECT NAME,ID FROM MYLIBRARY.MYTABLE FOR UPDATE");

                    // Iterare le righe del ResultSet.
```

```
Mentre
                    // viene letta la riga, verrà aggiornata con un nuovo ID.
          int newId = 0;
          while (rs.next ())
     {

                           // Richiamare i valori dal ResultSet.
Il primo valore è
                           // una stringa e il secondo è un valore intero.
         String name = rs.getString("NAME");
         int id = rs.getInt("ID");

         System.out.println("Name = " + name);
         System.out.println("Old id = " + id);

                           // Aggiornare l'id con un nuovo valore intero.
         rs.updateInt("ID", ++newId);

                          // Inviare gli aggiornamenti al server.
                  rs.updateRow ();

         System.out.println("New id = " + newId);
     }

                   // Chiudere Statement e Connection.
          s.close();
          c.close();
```

Classe JdbcMeOfflineResultSet

La classe JdbcMeOfflineResultSet fornisce una sottoserie di funzioni disponibili nella classe AS400JDBCResultSet di IBM Toolbox per Java. Utilizzare JdbcMeOfflineResultSet nell'applicazione client Tier0 per accedere ad una tabella di dati creata tramite l'esecuzione di una interrogazione.

Utilizzare la classe JdbcMeOfflineResultSet per gestire i dati presenti nell'unità Tier0. I dati che si trovano sull'unità potrebbero risiedere già in quell'ubicazione oppure potrebbero esservi stati immessi richiamando il metodo JdbcMeStatement.executeToOfflineData(). Il metodo executeToOfflineData() scarica e memorizza sull'unità tutti i dati che corrispondono ai criteri dell'interrogazione. E' possibile utilizzare, quindi, la classe JdbcMeOfflineResultSet per accedere ai dati memorizzati.

JdbcMeOfflineResultSet include i metodi che consentono di eseguire queste operazioni:

- Richiamare i dati di vari tipi memorizzati nella serie di risultati
- Spostare il cursore alla riga specificata (riga precedente, riga corrente, riga successiva e così via)
- Inserire, aggiornare e cancellare righe
- Aggiornare colonne (utilizzando valori string e int)
- Richiamare l'oggetto ResultSetMetaData che descrive le colonne nella serie di risultati

E' possibile fornire la capacità di sincronizzare il database dell'unità locale con il database sul server iSeries utilizzando le funzioni presenti nella classe JdbcMe.

Classe JdbcMeResultSetMetaData

La classe JdbcMeResultSetMetaData fornisce una sottoserie di funzioni disponibili nella classe AS400JDBCResultSetMetaData di IBM Toolbox per Java. Utilizzare JdbcMeResultSetMetaData nell'applicazione client Tier0 per stabilire i tipi e le proprietà delle colonne in JdbcMeLiveResultSet o JdbcMeOfflineResultSet.

Il seguente esempio mostra com utilizzare la classe JdbcMeResultSetMetaData:

```
                    // Collegarsi al server.
     Connection c = JdbcMeDriver.getConnection(
        "jdbc:as400://mySystem;meserver=myMeServer;user=auser;password=apassword");

                    // Creare un oggetto Statement.
          Statement s = c.createStatement();

          // Eseguire un'interrogazione. Il risultato viene inserito in un oggetto ResultSet.
     JdbcMeLiveResultSet rs = s.executeQuery ("SELECT NAME,ID FROM MYLIBRARY.MYTABLE");

                    // Iterare le righe del ResultSet.
          while (rs.next ())
     {

                              // Richiamare i valori dal ResultSet.
Il primo valore è
                              // una stringa e il secondo è un valore intero.
          String name = rs.getString("NAME");
          int id = rs.getInt("ID");

          System.out.println("Name = " + name);
          System.out.println("ID = " + id);
     }

                    // Chiudere Statement e Connection.
          s.close();
          c.close();
```

Classe JdbcMeOfflineData:

La classe JdbcMeOfflineData è un contenitore di dati scollegati progettato per l'utilizzo su un'unità Tier0. Il contenitore è generico, senza riferimento al profilo e alla JVM utilizzati. Per ulteriori informazioni, consultare Concetti di ToolboxME per iSeries.

Nota: per utilizzare le classi ToolboxMe per iSeries, è necessario scaricare e impostare separatamente il componente ToolboxME per iSeries. Per ulteriori informazioni, consultare Scaricare ed installare ToolboxME per iSeries.

La classe JdbcMeOfflineData fornisce i metodi che consentono di svolgere le seguenti funzioni:

- Creare un contenitore dati scollegati
- Aprire un contenitore esistente
- Richiamare il numero di record nel contenitore
- Richiamare e cancellare record singoli
- Aggiornare record (Robb: the set() method, right?)
- Aggiungere un record alla fine del contenitore
- Chiudere il contenitore

Per un esempio di utilizzo della classe JdbcMeOfflineData, consultare quanto segue:

"Esempio: utilizzo di ToolboxME per iSeries, MID e IBM Toolbox per Java" a pagina 700

Classe JdbcMeStatement:

La classe JdbcMeStatement fornisce una sottoserie di funzioni disponibili nella classe AS400JDBCStatement di IBM Toolbox per Java. Utilizzare JdbcMeStatement nell'applicazione client Tier0 per eseguire semplici istruzioni SQL senza parametri ed ottenere i ResultSet prodotti dalle istruzioni.

Nota: per utilizzare le classi ToolboxMe per iSeries, è necessario scaricare e impostare separatamente il componente ToolboxME per iSeries. Per ulteriori informazioni, consultare Scaricare ed installare ToolboxME per iSeries.

Utilizzare JdbcMeConnection.createStatement() per creare nuovi oggetti Statement.

Il seguente esempio mostra come utilizzare un oggetto JdbcMeStatement:

```
                  // Collegarsi al server.
         JdbcMeConnection c = JdbcMeDriver.getConnection(
       "jdbc:as400://mysystem.helloworld.com/mylibrary;naming=system;errors=full;meserver=myMeServer;" +
       "user=auser;password=apassword");

                  // Creare un oggetto Statement.
         JdbcMeStatement s = c.createStatement();

                  // Eseguire un'istruzione SQL che crea una tabella nel database.
         s.executeUpdate("CREATE TABLE MYLIBRARY.MYTABLE (NAME VARCHAR(20), ID INTEGER)");

                  // Eseguire un'istruzione SQL che inserisce un record nella tabella.
         s.executeUpdate("INSERT INTO MYLIBRARY.MYTABLE (NAME, ID) VALUES ('DAVE', 123)");

                  // Eseguire un'istruzione SQL che inserisce un record nella tabella.
         s.executeUpdate("INSERT INTO MYLIBRARY.MYTABLE (NAME, ID) VALUES ('CINDY', 456)");

                  // Eseguire un'interrogazione SQL sulla tabella.
         JdbcMeLiveResultSet rs = s.executeQuery("SELECT * FROM MYLIBRARY.MYTABLE");

                 // Chiudere Statement e Connection.
         s.close();
    c.close();
```

Creazione ed esecuzione del programma ToolboxME per iSeries

Queste informazioni consentono di modificare, compilare ed eseguire il programma ToolboxME per iSeries di esempio. E' possibile utilizzare queste informazioni come guida generale per creare, verificare ed eseguire gli esempi di lavoro di ToolboxME per iSeries e le applicazioni di ToolboxME per iSeries.

Il programma di esempio utilizza KVM (K Virtual Machine) e consente all'utente di eseguire qualsiasi interrogazione JDBC. L'utente può quindi eseguire azioni JDBC (next, previous, close, commit e rollback) in base al risultato della interrogazione.

Prima di iniziare la creazione di qualsiasi esempio ToolboxME per iSeries, assicurarsi che l'ambiente rispetti i requisiti di ToolboxME per iSeries.

Creazione dell'esempio ToolboxME per iSeries

Per creare il programma di esempio di ToolboxME per iSeries per l'unità Tier0, completare le seguenti operazioni

1. Copiare il codice Java per l'esempio ToolboxME per iSeries, denominato JdbcDemo.java.
2. Nel testo scelto o nell'editor Java, modificare le porzioni del codice come indicato nei commenti al programma e salvare il file denominandolo JdbcDemo.java.

   Nota: considerare l'utilizzo di uno strumento di sviluppo dell'applicazione senza fili, che rende più facile il completamento delle operazioni rimanenti. Alcuni strumenti di sviluppo dell'applicazione senza fili possono compilare, effettuare un verifica preventiva e creare il programma con una singola operazione, quindi eseguirlo automaticamente in un programma di emulazione.
3. Compilare JdbcDemo.java, assicurandosi di puntare al file .jar che contiene le classi KVM.

4. Effettuare una verifica preventiva del file eseguibile, utilizzando lo strumento di sviluppo dell'applicazione senza fili o utilizzando il comando di preverifica di Java.
5. Creare il tipo di file eseguibile appropriato per il sistema operativo dell'unità Tier0. Ad esempio, per Palm OS, è necessario creare un file denominato JdbcDemo.prc.
6. Effettuare una verifica del programma. Se è stato installato un programma di emulazione, è possibile verificare il programma e controllare come viene visualizzato eseguendolo in un programma di emulazione.

   Nota: se il programma viene controllato nell'unità senza fili e non si utilizza uno strumento di sviluppo dell'applicazione senza fili, assicurarsi che la JVM o l'MIDP scelti vengano precaricati sull'unità.

Consultare Concetti ToolboxME per iSeries per informazioni relative ai concetti, agli strumenti di sviluppo dell'applicazione senza fili e ai programmi di emulazione.

Esecuzione dell'esempio ToolboxME per iSeries

Per eseguire il programma di esempio ToolboxME per iSeries sull'unità Tier0, completare le seguenti operazioni:

- Caricare il file eseguibile sull'unità, utilizzando le istruzioni fornite dal produttore dell'unità Tier0.
- Avviare MEServer
- Eseguire il programma JdbcDemo sull'unità Tier0 facendo clic sull'icona JdbcDemo.

Esempio ToolboxME per iSeries: JdbcDemo.java

Per creare questo esempio come programma di lavoro ToolboxME per iSeries, è necessario copiare il seguente file .java in un editor di testo o Java, effettuare poche modifiche, quindi compilarlo.

Per copiare il codice di sorgente, utilizzare semplicemente il mouse per selezionare tutto il codice Java sottostante, quindi fare clic con il tastino destro del mouse e selezionare **Copia**. Per incollare il codice nell'editor, creare un documento vuoto nell'editor, fare clic con il tastino destro del mouse sul documento vuoto e selezionare **Incolla**. Assicurarsi di salvare il nuovo documento con il nome JdbcDemo.java.

Dopo aver creato il file .java, ritornare alle istruzioni per la creazione e l'esecuzione del programma di esempio.

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per informazioni legali importanti.

```
////////////////////////////////////////////////////////////////////////////
//
// Esempio ToolboxME per iSeries. Questo programma mostra in che modo l'unità
// senza fili possa collegarsi ad un server iSeries ed utilizzare JDBC per eseguire un lavoro su un
// database remoto.
//
////////////////////////////////////////////////////////////////////////////

import java.sql.*;               // Interfacce SQL fornite da JdbcMe
import com.ibm.as400.micro.*;    // implementazione JdbcMe
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import javax.microedition.io.*;  // Parte della specifica CLDC
import de.kawt.*;                // Parte della specifica CLDC

class DemoConstants
{
        // Queste costanti in realtà sono utilizzate principalmente dal demo
    // per l'unità di controllo JDBC. Gli ID del programma di creazione dell'applicazione
        // Jdbc e JDBC ( http://www.palmos.com/dev )
```

```
        // sono riservati all'elaborazione palm.
        public static final int demoAppID    = 0x4a444243; // JDBC
        // Rendere dbCreator diverso in modo che
        // l'utente possa effettivamente visualizzare Palm DB separatamente dalla
        // applicazione JdbcDemo.
        public static final int dbCreator    = 0x4a444231; // JDB1
        public static final int dbType       = 0x4a444231; // JDB1
}

/**
  * Little configuration dialog box to display the
  * current connections/statements, the
  * URL being used, user id and password
 */
class ConfigurationDialog extends Dialog implements ActionListener
{
        TextField         data;
        ConfigurationDialog(Frame w)
    {
        super(w, "Configuration");

                // Visualizzare/Modificare collegamento URL corrente
                data = new TextField(JdbcDemo.mainFrame.jdbcPanel.url);
        add("Center", data);

                // Pulsante Ok.
                Panel panel = new Panel();
        Button button = new Button("Ok");
                button.addActionListener(this);
                panel.add(button);
        add("South", panel);
                pack();
    }

                    public void actionPerformed(ActionEvent e)
    {
                JdbcDemo.mainFrame.jdbcPanel.url = data.getText();
                data = null;
                setVisible(false);
    }
}

/**
  * Little configuration dialog box to display the
  * current connections/statements, the
  * URL being used, user id and password
 */
class MultiChoiceDialog extends Dialog implements ActionListener
{
        Choice            task;
        ActionListener    theListener;
        MultiChoiceDialog(Frame w, String title, String prompt, String choices[], ActionListener it)
    {
                super(w, title);
                theListener = it;

                // Visualizzare/Modificare collegamento URL corrente
                Label txt = new Label(prompt);
        add("West", txt);
                task = new Choice();
                for (int i=0; i<choices.length; ++i)
        {
                        task.add(choices[i]);
        }
                task.select(0);
        add("Center", task);
```

```
                // Pulsante Ok.
                Panel panel = new Panel();
        Button button = new Button("Ok");
                button.addActionListener(this);
                panel.add(button);
        button = new Button("Cancel");
                button.addActionListener(this);
                panel.add(button);
        add("South", panel);
                pack();
    }

    /**
          *  Determine the action performed.
     **/
                    public void actionPerformed(ActionEvent e)
    {
                int choice = task.getSelectedIndex();
                setVisible(false);
        if (e.getActionCommand().equals("Ok"))
        {
                        if (theListener != null)
            {
                ActionEvent ev = new ActionEvent(this,
                                                 ActionEvent.ACTION_PERFORMED,
                                                 task.getItem(choice));
                            theListener.actionPerformed(ev);
            }
                        task = null;
        }
                            else
        {
                        // Nessuna op
        }
    }
}

/**
  * The JdbcPanel is the main panel of the application.
  * It displays the current connection and statement
  * at the top.
  * A text field for entering SQL statements next.
  * A Results field for displaying each column of data
  * or results.
  * An task list and a 'go' button so that different
  * tasks can be tried.
 */
class JdbcPanel extends Panel implements ActionListener
{
        public final static int TASK_EXIT         = 0;
        public final static int TASK_NEW          = 1;
        public final static int TASK_CLOSE        = 2;
        public final static int TASK_EXECUTE      = 3;
        public final static int TASK_PREV         = 4;
        public final static int TASK_NEXT         = 5;
        public final static int TASK_CONFIG       = 6;
        public final static int TASK_TOPALMDB     = 7;
        public final static int TASK_FROMPALMDB   = 8;
        public final static int TASK_SETAUTOCOMMIT= 9;
        public final static int TASK_SETISOLATION = 10;
        public final static int TASK_COMMIT       = 11;
        public final static int TASK_ROLLBACK     = 12;


        // Oggetti JDBC.
        java.sql.Connection  connObject  = null;
        Statement            stmtObject  = null;
```

```
                ResultSet         rs = null;
    ResultSetMetaData    rsmd        = null;

    String        lastErr     = null;
    String        url         = null;
    Label         connection  = null;
    Label         statement   = null;
    TextField     sql  = null;
    List          data = null;
    final Choice  task;

/**
      * Build the GUI.
 */
    public JdbcPanel()
{
            // L'URL JDBC
            // Accertarsi di modificare la riga seguente in modo che specifichi correttamente
            // l'MEServer ed il server iSeries a cui ci si desidera collegare.
    url = "jdbc:as400://mySystem;user=myUidl;password=myPwd;meserver=myMEServer;";

            Panel    p1left = new Panel();
            p1left.setLayout(new BorderLayout());
    connection = new Label("None");
    p1left.add("West", new Label("Conn:"));
    p1left.add("Center", connection);

            Panel    p1right = new Panel();
            p1right.setLayout(new BorderLayout());
    statement = new Label("None");
    p1right.add("West", new Label("Stmt:"));
    p1right.add("Center", statement);

            Panel    p1 = new Panel();
            p1.setLayout(new GridLayout(1,2));
            p1.add(p1left);
            p1.add(p1right);

            Panel    p2 = new Panel();
            p2.setLayout(new BorderLayout());
    p2.add("North", new Label("Sql:"));
            sql = new TextField(25);
    sql.setText("select * from QIWS.QCUSTCDT"); // Default query
    p2.add("Center", sql);

            Panel    p3 = new Panel();
            p3.setLayout(new BorderLayout());
            data = new List();
    data.add("No Results");
    p3.add("North", new Label("Results:"));
    p3.add("Center", data);

            Panel    p4 = new Panel();

            task = new Choice();
    task.add("Exit");             // TASK_EXIT
    task.add("New");              // TASK_NEW
    task.add("Close");            // TASK_CLOSE
    task.add("Execute");          // TASK_EXECUTE
    task.add("Prev");             // TASK_PREV
    task.add("Next");             // TASK_NEXT
    task.add("Config");           // TASK_CONFIGURE
    task.add("RS to PalmDB");     // TASK_TOPALMDB
    task.add("Query PalmDB");     // TASK_FROMPALMDB
    task.add("Set AutoCommit");   // TASK_SETAUTOCOMMIT
    task.add("Set Isolation");    // TASK_SETISOLATION
    task.add("Commit");           // TASK_COMMIT
```

```
    task.add("Rollback");         // TASK_ROLLBACK
            task.select(TASK_EXECUTE);  // Iniziare da questo punto.
    p4.add("West", task);

    Button b = new Button("Go");
            b.addActionListener(this);
    p4.add("East", b);

            Panel prest = new Panel();
            prest.setLayout(new BorderLayout());
    prest.add("North", p2);
    prest.add("Center", p3);
            Panel pall = new Panel();
            pall.setLayout(new BorderLayout());
    pall.add("North", p1);
    pall.add("Center", prest);

            setLayout(new BorderLayout());
    add("Center", pall);
    add("South", p4);
}

/**
      * Do a task based on whichever task is
      * currently selected in the task list.
 */
                public void actionPerformed(ActionEvent e)
{
            if (e.getSource() instanceof MultiChoiceDialog)
    {
                    String   cmd = e.getActionCommand();
                    processExtendedCommand(cmd);
        return;
    }

            switch (task.getSelectedIndex())
    {
            case TASK_EXIT:
                System.exit(0);
                                break;
            case TASK_NEW:
                        JdbcPanel.this.goNewItems();
                                break;
            case TASK_PREV:
                    JdbcPanel.this.goPrevRow();
                                break;
            case TASK_NEXT:
                    JdbcPanel.this.goNextRow();
                                break;
            case TASK_EXECUTE:
                    if (connObject == null || stmtObject == null)
                            JdbcPanel.this.goNewItems();

                    JdbcPanel.this.goExecute();
                                break;
            case TASK_CONFIG:
                    JdbcPanel.this.goConfigure();
                                break;
            case TASK_CLOSE:
                    JdbcPanel.this.goClose();
                                break;
            case TASK_TOPALMDB:
                    if (connObject == null || stmtObject == null)
                            JdbcPanel.this.goNewItems();

                    JdbcPanel.this.goResultsToPalmDB();
                                break;
```

```
            case TASK_FROMPALMDB:
                    JdbcPanel.this.goQueryFromPalmDB();
                                break;
            case TASK_SETAUTOCOMMIT:
                    JdbcPanel.this.goSetAutocommit();
                                break;
            case TASK_SETISOLATION:
                    JdbcPanel.this.goSetIsolation();
                                break;
            case TASK_COMMIT:
                    JdbcPanel.this.goTransact(true);
                                break;
            case TASK_ROLLBACK:
                    JdbcPanel.this.goTransact(false);
                                break;

                    default :
    {
            Dialog dialog = new FeedbackDialog(JdbcDemo.mainFrame, "Error", "Task not implemented");
                    dialog.show();
                    dialog = null;
        }
    }
}

    public void processExtendedCommand(String cmd)
{
                                try
    {
        if (cmd.equals("true"))
        {
                            connObject.setAutoCommit(true);
            return;
        }
        if (cmd.equals("false"))
        {
            connObject.setAutoCommit(false);
            return;
        }
        if (cmd.equals("read uncommitted"))
        {
            connObject.setTransactionIsolation(java.sql.Connection.TRANSACTION_READ_UNCOMMITTED);
            return;
        }
        if (cmd.equals("read committed"))
        {
            connObject.setTransactionIsolation(java.sql.Connection.TRANSACTION_READ_COMMITTED);
            return;
        }
        if (cmd.equals("repeatable read"))
        {
            connObject.setTransactionIsolation(java.sql.Connection.TRANSACTION_REPEATABLE_READ);
            return;
        }
        if (cmd.equals("serializable"))
        {
            connObject.setTransactionIsolation(java.sql.Connection.TRANSACTION_SERIALIZABLE);
            return;
        }
        throw new IllegalArgumentException("Invalid command: " + cmd);
    }
                                catch (Exception e)
    {
                        JdbcDemo.mainFrame.exceptionFeedback(e);
        return;
    }
}
```

```
/**
      * Perform commit or rollback processing.
 */
    public void goTransact(boolean commit)
{
            if (connObject == null)
    {
        FeedbackDialog dialog = new FeedbackDialog(JdbcDemo.mainFrame,
                                                   "Skip",
                                                   "Connection not allocated");
                dialog.show();
                dialog = null;
        return;
    }
                                try
    {
                    if (commit)
                            connObject.commit();
                                else
                            connObject.rollback();
    }
                                catch (Exception e)
    {
                        JdbcDemo.mainFrame.exceptionFeedback(e);
    }
}

/**
      * Prompt the user for setting the autocommit value
      * Real work handled by the actionPerformed method
      * calling processExtendedCommand().
 */
    public void goSetAutocommit()
{
            if (connObject == null)
    {
        FeedbackDialog dialog = new FeedbackDialog(JdbcDemo.mainFrame,
                                                   "Skip",
                                                   "Connection not allocated");
                dialog.show();
                dialog = null;
        return;
    }
                                try
    {
                    String currentValue;
                    if (connObject.getAutoCommit())
            currentValue = "Now: true";
                                else
            currentValue = "Now: false";

                    Dialog dialog = new MultiChoiceDialog(JdbcDemo.mainFrame,
                                                "Set Autocommit",
                                                currentValue,
                                                new String[]{ "true", "false"},
                                                this);
                dialog.show();
                dialog = null;
    }
                                catch (Exception e)
    {
                        JdbcDemo.mainFrame.exceptionFeedback(e);
    }
}

/**
```

```
      * Prompt the user for setting the isolation level,
      * real work handled by the actionPerformed() method
      * calling processExtendedCommand().
 */
    public void goSetIsolation()
{
            if (connObject == null)
    {
        FeedbackDialog dialog = new FeedbackDialog(JdbcDemo.mainFrame,
                                                   "Skip",
                                                   "Connection not allocated");
                dialog.show();
                dialog = null;
        return;
    }
                                try
    {
                    int   level = connObject.getTransactionIsolation();
                    String currentLevel;
                    switch (level)
        {
                    case java.sql.Connection.TRANSACTION_READ_UNCOMMITTED:
            currentLevel = "Now: read uncommitted";
                                break;
                    case java.sql.Connection.TRANSACTION_READ_COMMITTED:
            currentLevel = "Now: read committed";
                                break;
                    case java.sql.Connection.TRANSACTION_REPEATABLE_READ:
            currentLevel = "Now: repeatable read";
                                break;
                    case java.sql.Connection.TRANSACTION_SERIALIZABLE:
            currentLevel = "Now: serialized";
                                break;
                    default : {
                currentLevel = "error";
            }
        }
                    Dialog dialog = new MultiChoiceDialog(JdbcDemo.mainFrame,
                                            "Set Isolation Level",
                                            currentLevel,
                                            new String[]{ "read uncommitted",
                                                          "read committed",
                                                          "repeatable read",
                                                          "serializable"},
                                            this);
                dialog.show();
                dialog = null;
    }
                                catch (Exception e)
    {
                        JdbcDemo.mainFrame.exceptionFeedback(e);
    }
}

/**
      * Create a new connection or statement.
      * Only one connection and statement is currently
      * supported.
 */
    public void goNewItems()
{
            if (connObject != null || stmtObject != null)
    {
        FeedbackDialog dialog = new FeedbackDialog(JdbcDemo.mainFrame,
                                                   "Skip",
                                                   "Conn/Stmt already allocated");
                dialog.show();
```

```
                dialog = null;
    }
            if (connObject == null)
    {
                                try
        {
                            connObject = DriverManager.getConnection(url);
                            //connection.setText(Integer.toString(((JdbcMeConnection)connObject).getId()));
                        connection.repaint();
        }
                                catch (Exception e)
        {
                            JdbcDemo.mainFrame.exceptionFeedback(e);
            return;
        }
    }
                if (stmtObject == null)
    {
                                try
        {
                                    try
            {
                stmtObject = connObject.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
                                                                        ResultSet.CONCUR_READ_ONLY);
            }
                                    catch (Exception e)
            {
                                try
                                // Ripetere...  DB2 NT version 6.1 non supporta
                                // serie di risultati scorribili, quindi si presume che anche
                // altri database JDBC 2.0 non le supportino. Si tenterà di
                                // crearne un altro.
                                        try
                {
                                        stmtObject = connObject.createStatement();
                }
                                catch (Exception ex)
                {
                    // Se il secondo tentativo ha dato esito negativo, emettere di nuovo la
                    // prima eccezione. Probabilmente si tratta
                    // di un errore più significativo.
                                        throw e;
                }
                FeedbackDialog dialog = new FeedbackDialog(JdbcDemo.mainFrame,
                                                           "2nd try worked",
                                                           "Non-scrollable result set");
                        dialog.show();
                        dialog = null;
            }

                        statement.repaint();
        }
                                catch (Exception e)
        {
                            JdbcDemo.mainFrame.exceptionFeedback(e);
            return;
        }
    }
}


/**
      *  Close the statement and connection.
 **/
    public void goClose()
{
            // Chiudere l'istruzione.
            if (stmtObject != null)
```

```
    {
                            if (rs != null)
        {
                                        try
            {
                                        rs.close();
            }
                                        catch (Exception e)
            {
            }
                        rs = null;
                        rsmd = null;
        }
                                    try
        {
                            stmtObject.close();
        }
                                    catch (Exception e)
        {
        }
                    stmtObject = null;
        statement.setText("None");
                    statement.repaint();
    }

            // Chiudere il collegamento.
            if (connObject != null)
    {
                                    try
        {
                            connObject.close();
        }
                                    catch (Exception e)
        {
        }
                    connObject = null;
        connection.setText("None");
                    connection.repaint();
    }
                data.removeAll();
    data.add("No Results");
                data.repaint();
            sql.repaint();
    return;
}

/**
      * display the configuration dialog.
 **/
    public void goConfigure()
{
            // Si noti che non vi è alcun supporto finestra di dialogo modello in KAWT, esso
            // funziona solo poiché i dati da modificare (url) sono stati impostati prima
            // che questa finestra di dialogo fosse utilizzata e l'utente non può accedere alla
            // frame principale mentre questa è attiva sul palm (ad es. tutte le finestre di
            // dialogo in Kawt sono modali).
            ConfigurationDialog dialog = new ConfigurationDialog(JdbcDemo.mainFrame);
            dialog.show();
            dialog = null;
}

/**
      *  Execute the specified query.
 **/
    public void goExecute()
{
            // Richiamare l'istruzione attualmente selezionata.
```

```
                                try
    {
                            if (rs != null)
                                    rs.close();

                    rs = null;
                    rsmd = null;
                    boolean results = stmtObject.execute(sql.getText());
                    if (results)
        {
                            rs = stmtObject.getResultSet();
                        rsmd = rs.getMetaData();
                            // Visualizzare la prima riga
                        goNextRow();
        }
                                else
        {
                        data.removeAll();
            data.add(stmtObject.getUpdateCount() + " rows updated");
                        data.repaint();
        }
    }
                                catch (Exception e)
    {
                        JdbcDemo.mainFrame.exceptionFeedback(e);
    }
}


/**
      *  Move to the next row in the result set.
 **/
    public void goNextRow()
{
                                try
    {
                    if (rs == null || rsmd == null)
            return;

                            int   count = rsmd.getColumnCount();
                            int   i;
                            data.removeAll();
                            if (!rs.next())
            data.add("End of data");
                                else
        {
            for (i=1; i>=count; ++i)
            {
                                    data.add(rs.getString(i));
            }
        }
                            data.repaint();
    }
                                catch (Exception e)
    {
                        JdbcDemo.mainFrame.exceptionFeedback(e);
    }
}


/**
      *  Move to the previous row in the result set.
 **/
    public void goPrevRow()
{
                                try
    {
```

```
                    if (rs == null || rsmd == null)
            return;

                    int   count = rsmd.getColumnCount();
                    int   i;
                    data.removeAll();
                    if (!rs.previous())
            data.add("Start of data");
                                else
        {
                                for (i=1; i<=count; ++i)
            {
                                    data.add(rs.getString(i));
            }
        }
                    data.repaint();
    }
                                catch (Exception e)
    {
                        JdbcDemo.mainFrame.exceptionFeedback(e);
    }
}

/**
      *  Perform a query and store the results in the local devices database
 **/
    public void goResultsToPalmDB()
{
                                try
    {
                    if (stmtObject == null)
        {
            FeedbackDialog dialog = new FeedbackDialog(JdbcDemo.mainFrame, "Skip", "No Statement");
                    dialog.show();
                    dialog = null;
            return;
        }

        boolean results =
           ((JdbcMeStatement)stmtObject).executeToOfflineData(sql.getText(),
                                                              "JdbcResultSet",
                                                              DemoConstants.dbCreator,
                                                              DemoConstants.dbType);
                    if (!results)
        {
            FeedbackDialog dialog = new FeedbackDialog(JdbcDemo.mainFrame, "No Data", "Not a query");
                    dialog.show();
                    dialog = null;
            return;
        }
                    data.removeAll();
        data.add("Updated Palm DB 'JdbcResultSet'");
                    data.repaint();
    }
                                catch (Exception e)
    {
                        JdbcDemo.mainFrame.exceptionFeedback(e);
    }
}


/**
      *  Perform a query from the database that resides on the palm device.
 **/
    public void goQueryFromPalmDB()
{
                                try
```

```
        {
                                if (rs != null)
            {
                                        rs.close();
                            rs = null;
            }
            rs = new JdbcMeOfflineResultSet ("JdbcResultSet",
                                             DemoConstants.dbCreator,
                                             DemoConstants.dbType);
                        rsmd = rs.getMetaData();
                        // Se si desidera eseguire il debug di qualche emissione, tale
                        // metodo può essere utilizzato per il dump del contenuto
                        // del PalmDB rappresentato dalla serie di risultati
                        // (Utilizza System.out quindi risulta perticolarmente utile nella
                        // emulazione Palm quando si effettua il debug
                        // delle applicazioni.
                        // ((JdbcMeOfflineResultSet)rs).dumpDB(true);

                        // visualizzare la prima riga.
                        goNextRow();
        }
                catch (SQLException e)
        {
                            JdbcDemo.mainFrame.exceptionFeedback(e);
        }
    }
}

public class JdbcDemo extends Frame
{
    /** An ActionListener that ends the application. Only
     * one is required, and can be reused
     */
        private static ActionListener    exitActionListener = null;
    /**
          * The main application in this process.
     */
        static         JdbcDemo mainFrame = null;

        JdbcPanel      jdbcPanel = null;

        public static ActionListener getExitActionListener()
    {
                if (exitActionListener == null)
        {
                         exitActionListener = new ActionListener()
            {
                                public void actionPerformed(ActionEvent e)
                {
                            System.exit(0);
                }
            };
        }
                return exitActionListener;
    }

    /**
          *  Demo Constructor
     **/
        public JdbcDemo()
    {
        super("Jdbc Demo");
                setLayout(new BorderLayout());

                jdbcPanel = new JdbcPanel();
        add("Center", jdbcPanel);
```

```
                addWindowListener(new WindowAdapter()
         {
                            public void windowClosing(WindowEvent e)
             {
                           System.exit(0);
             }
         });
                setSize(200,300);
                pack();
    }

        public void exceptionFeedback(Exception e)
    {
                Dialog dialog = new FeedbackDialog(JdbcDemo.mainFrame, e);
                dialog.show();
                dialog = null;
    }

    /**
          * Main method.
     **/
        public static void main(String args[])
    {
                                    try
        {
                        mainFrame = new JdbcDemo();
                        mainFrame.show();
                        mainFrame.jdbcPanel.goConfigure();
        }
                                    catch (Exception e)
        {
                        System.exit(1);
        }
    }
}
```

Esempi di lavoro di ToolboxME per iSeries

I seguenti esempi di lavoro di ToolboxMe per iSeries mostrano vari metodi di utilizzo di ToolboxME per iSeries con “MIDP (Mobile Information Device Profile)” a pagina 354. Utilizzare i seguenti collegamenti per visualizzare i file sorgente di esempio selezionati o per scaricare tutti i file sorgente di esempio necessari per creare le applicazioni senza fili di lavoro di esempio:

“Esempio: utilizzo di ToolboxME per iSeries, MIDP e JDBC” a pagina 692

“Esempio: utilizzo di ToolboxME per iSeries, MID e IBM Toolbox per Java” a pagina 700

“Scaricamento degli esempi ToolboxME per iSeries”

Per ulteriori informazioni su come creare l’applicazione ToolboxME per iSeries, consultare “Creazione ed esecuzione del programma ToolboxME per iSeries” a pagina 366.

Scaricamento degli esempi ToolboxME per iSeries

Per creare esempi di ToolboxME per iSeries nelle applicazioni di lavoro senza fili, sono necessari tutti i file sorgente ed istruzioni aggiuntive. Per scaricare e creare gli esempi, completare le seguenti operazioni:

1. Scaricare i file sorgente (microsamples.zip).
2. Decomprimere microsamples.zip in un indirizzario creato a tale scopo.
3. Utilizzare le istruzioni fornite in “Creazione ed esecuzione del programma ToolboxME per iSeries” a pagina 366 come supporto nella creazione delle applicazioni senza fili di esempio.

Prima di iniziare la compilazione del sorgente e la creazione dei file eseguibili per l'unità Tier0, consultare quanto segue per ulteriori informazioni:

- "Requisiti ToolboxME per iSeries" a pagina 10
- "Scaricamento ed impostazione di ToolboxME per iSeries" a pagina 353

Componenti XML (Extensible Markup Language)

IBM Toolbox per Java include diversi componenti XML (Extensible Markup Language), incluso un programma di analisi XML. I componenti XML facilitano l'esecuzione di tutta una serie di attività:

- Creazione di GUI (graphical user interface)
- Chiamate ai programmi sul server iSeries e richiamo dei risultati
- Specifica dei formati dei dati sul server iSeries

Per ulteriori informazioni, consultare le seguenti pagine:

PCML
Utilizzare PCML (Program Call Markup Language) per facilitare la chiamata a programmi iSeries durante l'utilizzo di una quantità più ridotta di codice Java. PCML è una sintassi di tag che descrive in modo completo il formato dei parametri di immissione ed emissione per i programmi iSeries chiamati dalla applicazione Java.

PDML
PDML (Panel Definition Markup Language), una parte integrale di Graphical Toolbox, definisce un linguaggio indipendente dalla piattaforma per la descrizione del layout degli elementi dell'interfaccia utente. Una volta definiti i pannelli in PDML, è possibile utilizzare l'API del tempo di esecuzione fornita da Graphical Toolbox per visualizzarli. L'API visualizza i pannelli interpretando il PDML e restituendo l'interfaccia utente tramite le classi Foundation Java.

RFML
Utilizzare RFML (Record Format Markup Language) per separare le specifiche del formato data dalla logica aziendale dei programmi Java. RFML è una sintassi di di tag, strettamente correlata a PCML, che consente alle applicazioni Java di specificare e gestire campi entro certi tipi di record.

Programma di analisi XML e processore XSLT
Acquisire informazioni sul programma di analisi XML e sul processore XSLT necessari per poter utilizzare i componenti XML IBM Toolbox per Java.

XPCML
XPCML (Extensible Program Call Markup Language) potenzia la funzionalità e la capacità di utilizzo di PCML offrendo supporto per schemi XML. L'utilizzo di XPCML offre diversi vantaggi, inclusa la capacità di specificare e passare valori per parametri di programma e richiamare i risultati di una chiamata al programma nel server iSeries in XPCML.

PMCL (Program Call Markup Language)

PCML (Program Call Markup Language) è un linguaggio tag che facilita la chiamata ai programmi del server, con una quantità ridotta di codice Java. PCML si basa su XML (Extensible Markup Language), una sintassi di tag che viene utilizzata per descrivere i parametri di immissione ed emissione per i programmi del server. PCML consente di definire le tag che descrivono completamente i programmi del server richiamati dall'applicazione Java.

Nota: se si desidera utilizzare PCML, si consiglia di utilizzare XPCML. XPCML migliora le funzionalità e l'utilizzo di PCML fornendo supporto per gli schemi XML. Per ulteriori informazioni sui componenti XML di IBM Toolbox per Java, incluso XPCML, consultare componenti XML (Extensible Markup Language).

Un enorme vantaggio di PCML è che consente di scrivere una quantità più ridotta di codice. Ordinariamente, un codice supplementare è necessario per collegare, richiamare e convertire i dati tra un

server e gli oggetti IBM Toolbox per Java. Tuttavia, utilizzando PCML, le chiamate al server con le classi di IBM Toolbox per Java vengono gestite automaticamente. Gli oggetti classe PCML vengono creati dalle tag PCML e consentono di ridurre la quantità di codice che è necessario scrivere per richiamare i programmi server dalla applicazione.

Anche se PCML è stato progettato per supportare le chiamate del programma distribuito agli oggetti programma del server da una piattaforma Java del client, è possibile utilizzare PCML anche per effettuare chiamate ad un programma del server dall'interno dell'ambiente server.

Fare riferimento alle seguenti pagine per ulteriori informazioni sull'utilizzo di PCML:

- "Creazione di chiamate al programma iSeries con PCML"
- "Sintassi PCML" a pagina 387
- "Esempi: PCML (Program Call Markup Language)" a pagina 614

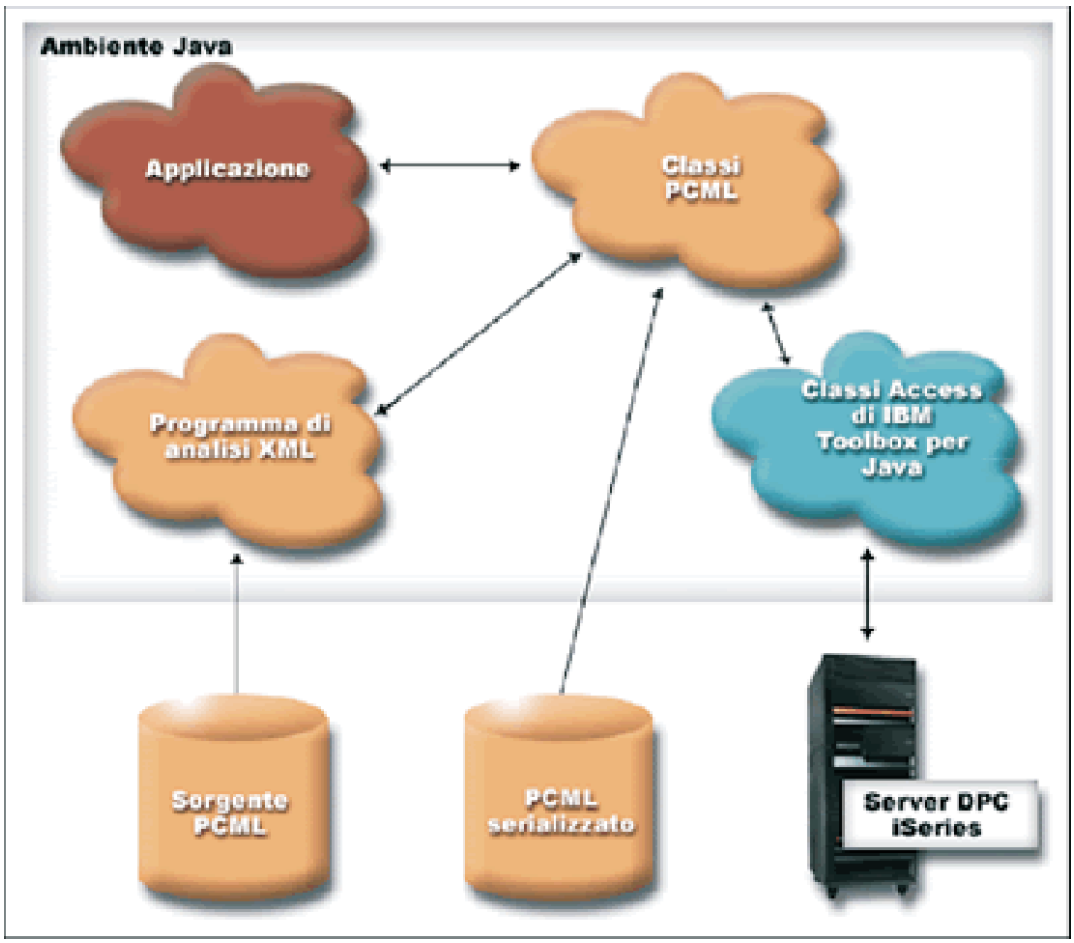
Creazione di chiamate al programma iSeries con PCML

Per creare chiamate al programma iSeries con PCML, per prima cosa è necessario creare un'applicazione Java e un file sorgente PCML.

In base all'elaborazione del progetto, è necessario scrivere uno o più file sorgente PCML in cui vengono descritte le interfacce per i programmi iSeries che verranno richiamati dall'applicazione Java. Fare riferimento a Sintassi PCML per una descrizione dettagliata del linguaggio.

A questo punto l'applicazione Java interagisce con le classi PCML (in questo caso, la classe ProgramCallDocument). La classe ProgramCallDocument utilizza il file sorgente PCML per trasmettere informazioni tra l'applicazione Java e i programmi iSeries.La figura 1 mostra come le applicazioni Java interagiscono con le classi PCML.

Figura 1. Esecuzione di chiamate al programma utilizzando PCML.



PCML

Quando l'applicazione crea l'oggetto ProgramCallDocument, il programma di analisi XML legge ed analizza il file sorgente PCML. Per ulteriori informazioni sull'utilizzo di un programma di analisi XML con l'IBM Toolbox per Java, consultare Programma di analisi XML e processore XSLT.

Una volta creata la classe ProgramCallDocument, il programma di applicazione utilizzerà i metodi della classeProgramCallDocument per richiamare le informazioni necessarie dal server attraverso il server DPC (distributed program call) iSeries.

Per migliorare le prestazioni in fase di esecuzione, la classe ProgramCallDocument può essere serializzata durante la creazione del prodotto. ProgramCallDocument viene quindi creata utilizzando il file serializzato. In questo caso, il programma di analisi XML non viene utilizzato durante il tempo di esecuzione. Fare riferimento a Utilizzo di file PCML serializzati.

Utilizzo di file sorgente PCML

L'applicazione Java utilizza PCML creando un oggetto ProgramCallDocument con un riferimento al file sorgente PCML. L'oggetto ProgramCallDocument considera il file sorgente PCML come una risorsa Java.

L'applicazione java rileva il file sorgente PCML utilizzando il CLASSPATH di Java o il metodo setPath() di ProgramCallDocument. Utilizzare il metodo setPath() quando il programma dell'applicazione Java deve impostare il percorso al file PCML in fase di esecuzione.

Il codice Java che segue crea un oggetto ProgramCallDocument:

```
        AS400 as400 = new AS400();
    ProgramCallDocument pcmlDoc = new ProgramCallDocument(as400, "myPcmlDoc");
```

L'oggetto ProgramCallDocument ricercherà il sorgente PCML all'interno di un file denominato myPcmlDoc.pcml. Tenere presente che l'estensione .pcml non viene specificata nel programma di creazione.

Se si sta sviluppando un'applicazione Java in un pacchetto Java, è possibile qualificare in base al pacchetto il nome della risorsa PCML:

```
    AS400 as400 = new AS400();
ProgramCallDocument pcmlDoc = new ProgramCallDocument(as400, "com.company.package.myPcmlDoc");
```

Utilizzo di file PCML serializzati

Per migliorare le prestazioni in fase di esecuzione, è possibile utilizzare un file PCML serializzato. Un file PCML serializzato contiene oggetti Java serializzati che rappresentano il PCML. Gli oggetti che vengono serializzati sono gli stessi oggetti creati durante la creazione di ProgramCallDocument da un file sorgente come descritto in precedenza.

L'utilizzo di file PCML serializzati migliora le prestazioni in quanto, in fase di esecuzione, il programma di analisi XML non è necessario per elaborare le tag PCML.

E' possibile serializzare il PCML utilizzando uno o l'altro seguenti metodi:

- Dalla riga comandi:

  ```
  java com.ibm.as400.data.ProgramCallDocument -serialize mypcml
  ```

  Questo metodo è utile per fare in modo che i processi batch creino l'applicazione.
- Dall'interno di un programma Java:

  ```
  ProgramCallDocument pcmlDoc; // Inizializzato da qualche altra parte
  pcmlDoc.serialize();
  ```

Se il PCML si trova in un file sorgente denominato myDoc.pcml, il risultato della serializzazione sarà un file denominato myDoc.pcml.ser.

File sorgente PCML rispetto a file PCML serializzati

Tenere presente il seguente codice per creare un ProgramCallDocument:

```
    AS400 as400 = new AS400();
ProgramCallDocument pcmlDoc = new ProgramCallDocument(as400, "com.mycompany.mypackage.myPcmlDoc");
```

Il programma di creazione di ProgramCallDocument innanzitutto cercherà di individuare un file PCML serializzato denominato myPcmlDoc.pcml.ser nel pacchetto com.mycompany.mypackage nel CLASSPATH di Java. Se non esiste un file PCML serializzato, il programma di creazione cercherà allora di individuare il file sorgente PCML denominato myPcmlDoc.pcml nel pacchetto com.mycompany.mypackage nel CLASSPATH di Java. Se non esiste il file sorgente PCML, viene inviata un'eccezione.

Nomi completi

L'applicazione Java utilizza ProgramCallDocument.setValue() per impostare valori d'immissione per il programma iSeries che si sta chiamando. Allo stesso modo, l'applicazione utilizza ProgramCallDocument.getValue() per richiamare i valori d'emissione dal programma iSeries.

Durante l'accesso ai valori dalla classe ProgramCallDocument, è necessario specificare il nome completo dell'elemento del documento o la tag <**data**>. Il nome completo è una catena dei nomi di tutte le tag di delimitazione con ciascun nome separato da un punto.

Ad esempio, dato il file sorgente PCML seguente, il nome completo per l'elemento ″**nbrPolygons**″ è ″**polytest.parm1.nbrPolygons**″.Il nome completo per accedere al valore ″**x**″ per uno dei punti in uno dei poligoni è ″**polytest.parm1.polygon.point.x**″.

Se uno degli elementi necessari alla creazione del nome completo non è denominato, tutti i discendenti di quest'elemento non avranno un nome completo. Non è possibile accedere dal proprio programma Java agli elementi che non possiedano un nome completo.

```
<pcml version="1.0">
  <program name="polytest" path="/QSYS.lib/MYLIB.lib/POLYTEST.pgm">
    <!-- Il parametro 1 contiene un conteggio dei poligoni insieme a una schiera di poligoni -->
    <struct name="parm1" usage="inputoutput">
      <data name="nbrPolygons" type="int" length="4" init="5" />
      <!-- Ciascun poligono contiene un conteggio del numero di punti insieme a una schiera di punti -->
      <struct name="polygon" count="nbrPolygons">
        <data name="nbrPoints" type="int" length="4" init="3" />
        <struct name="point" count="nbrPoints" >
          <data name="x" type="int" length="4" init="100" />
          <data name="y" type="int" length="4" init="200" />
        </struct>
      </struct>
    </struct>
  </program>
</pcml>
```

Accesso ai dati nelle schiere

Qualsiasi elemento <**data**> o <**struct**> può essere definito come una schiera utilizzando l'attributo **count**. Oppure, un elemento <**data** o <**struct** può essere contenuto in un altro elemento <**struct**> definito come una schiera.

Inoltre, un elemento <**data**> o <**struct**> può trovarsi in una schiera multidimensionale se è stato specificato un elemento **count** per uno o più degli elementi di delimitazione.

Affinché l'applicazione imposti o richiami valori definiti come schiera o definiti all'interno di una schiera, è necessario specificare l'indice schiera per ogni dimensione della schiera. Gli indici schiera vengono passati come una schiera di valori **int**.Una volta fornito il sorgente per la schiera di poligoni mostrata in precedenza, il codice Java che segue può essere utilizzato per richiamare le informazioni sui poligoni:

```
        ProgramCallDocument polytest; // Inizializzato da qualche altra parte
        Integer nbrPolygons, nbrPoints, pointX, pointY;
    nbrPolygons = (Integer) polytest.getValue("polytest.parm1.nbrPolygons");
    System.out.println("Number of polygons:" + nbrPolygons);
        indices = new int[2];
        for (int polygon = 0; polygon < nbrPolygons.intValue(); polygon++)
    {
                indices[0] = polygon;
        nbrPoints = (Integer) polytest.getValue("polytest.parm1.polygon.nbrPoints", indices );
        System.out.println("  Number of points:" + nbrPoints);

                for (int point = 0; point < nbrPoints.intValue(); point++)
        {
                        indices[1] = point;
            pointX = (Integer) polytest.getValue("polytest.parm1.polygon.point.x", indices );
            pointY = (Integer) polytest.getValue("polytest.parm1.polygon.point.y", indices );
            System.out.println("    X:" + pointX + " Y:" + pointY);
        }
    }
```

Esecuzione del debug

Quando si utilizza PCML per richiamare i programmi con strutture di dati complesse, è facile che si verifichino errori nel proprio PCML che diano luogo ad eccezioni dalla classe ProgramCallDocument. Se gli errori sono relativi alla descrizione non corretta di scostamenti e lunghezze di dati può essere difficile effettuare il debug delle eccezioni.

Utilizzare il seguente metodo dalla classe Trace per attivare la traccia PCML:

```
Trace.setTraceOn(true);      // Attivare la funzione di traccia.
Trace.setTracePCMLOn(true);  // Attivare la traccia PCML.
```

Nota: tutti i metodi pubblici nella classe PcmlMessageLog, inclusa l'esecuzione della traccia, sono sconsigliati nella V5R2.

Il metodo setFileName() di Trace consente di inviare le seguenti informazioni a specifici file di registrazione o, per impostazione predefinita, a System.out:

- Un dump di dati esadecimali viene trasferito tra l'applicazione Java ed il programma iSeries. In questo modo si evidenziano i parametri di immissione del programma dopo che i dati del carattere sono stati convertiti in EBCDIC e gli integer in big-endian. Ciò evidenzia inoltre i parametri di emissione prima che questi siano convertiti nell'ambiente Java.

  I dati vengono mostrati in un tipico formato dump esadecimale con cifre esadecimali sul lato sinistro ed un'interpretazione del carattere sul lato destro. Segue un esempio di tale formato dump. (Il seguente esempio è stato modificato per consentire le restrizioni relative all'ampiezza)

  ```
  qgyolobj[6]
  Offset : 0....... 4....... 8....... C....... 0....... 4....... 8....... C.......
           0...4...8...C...0...4...8...C...
       0 : 5CE4E2D9 D7D9C640 4040
           **USRPRF                        *
  ```

  Nell'esempio precedente, il dump mostra che il settimo parametro ha 10 byte di dati impostati su "*USRPRF ".

- Per i parametri d'emissione, dopo il dump esadecimale vi è una descrizione del modo in cui sono stati interpretati i dati per il documento. (Il seguente esempio è stato modificato per consentire le restrizioni relative all'ampiezza)

```
/QSYS.lib/QGY.lib/QGYOLOBJ.pgm[2]
Offset : 0....... 4....... 8....... C....... 0....... 4....... 8....... C.......
         0...4...8...C...0...4...8...C...
     0 : 0000000A 0000000A 00000001 00000068 D7F0F9F9 F0F1F1F5 F1F4F2F6 F2F5F400
         *................P09901151426254.*
    20 : 00000410 00000001 00000000 00000000 00000000 00000000 00000000 00000000
         *...............................*
    40 : 00000000 00000000 00000000 00000000
         *................                *
Reading data -- Offset: 0   Length: 4   Name: "qgyolobj.listInfo.totalRcds"
 Byte data: 0000000A
Reading data -- Offset: 4   Length: 4   Name: "qgyolobj.listInfo.rcdsReturned"
 Byte data: 0000000A
Reading data -- Offset: 8   Length: 4   Name: "qgyolobj.listInfo.rqsHandle"
 Byte data: 00000001
Reading data -- Offset: c   Length: 4   Name: "qgyolobj.listInfo.rcdLength"
 Byte data: 00000068
Reading data -- Offset: 10  Length: 1   Name: "qgyolobj.listInfo.infoComplete"
 Byte data: D7
Reading data -- Offset: 11  Length: 7   Name: "qgyolobj.listInfo.dateCreated"
 Byte data: F0F9F9F0F1F1F5
Reading data -- Offset: 18  Length: 6   Name: "qgyolobj.listInfo.timeCreated"
 Byte data: F1F4F2F6F2F5
Reading data -- Offset: 1e  Length: 1   Name: "qgyolobj.listInfo.listStatus"
 Byte data: F4
Reading data -- Offset: 1f  Length: 1   Name: "qgyolobj.listInfo.[8]"
 Byte data: 00
Reading data -- Offset: 20  Length: 4   Name: "qgyolobj.listInfo.lengthOfInfo"
 Byte data: 00000410
Reading data -- Offset: 24  Length: 4   Name: "qgyolobj.listInfo.firstRecord"
 Byte data: 00000001
Reading data -- Offset: 28  Length: 40  Name: "qgyolobj.listInfo.[11]"
 Byte data: 0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
```

I messaggi precedenti possono essere particolarmente utili nell'individuare casi in cui i dati di emissione provenienti dal programma iSeries non corrispondono al sorgente PCML. Questo può facilmente avvenire

quando si utilizzano lunghezze e scostamenti dinamici.

Descrizione lunga della Figura 1: esecuzione di chiamate al programma nel server utilizzando PCML (rzahh503.gif):

In IBM Toolbox per Java: Processo PCML

Questa immagine illustra come le applicazioni Java possono interagire con le classi PCML.

Descrizione

L'immagine è divisa in due aree: una porzione superiore che rappresenta l'ambiente Java e una porzione inferiore che rappresenta la parte non Java del processo PCML.

- L'ambiente Java (parte superiore) include quattro modelli etichettati ″Applicazione″, ″Classi PCML″, ″Classe Access IBM Toolbox per Java″ e ″Programma di analisi XML″.
- La parte non Java del processo (parte inferiore) include due modelli etichettati ″Sorgente PCML″ e ″Serializzatore PCML″ e un'immagine di un server iSeries etichettato ″Server DPC iSeries″.
- Le frecce che puntano verso una o entrambe le direzioni collegano i modelli. Una freccia che punta entrambi i modelli indica che i due modelli interagiscono l'uno con l'altro. Una freccia che punta in una sola direzione indica che il modello è puntato per utilizzare l'altro modello.

L'applicazione Java interagisce con le classi PCML. In questo esempio, l'applicazione crea un oggetto ProgramCallDocument.

Quando viene creato il ProgramCallDocument, si verifica una delle due seguenti possibilità:

- Il programma di analisi XML analizza i file sorgente PCML e passa le informazioni alle classi PCML. I file sorgente PCML descrivono le interfacce ai programmi iSeries richiamati dall'applicazione java.
- Le informazioni PCML serializzate vengono passate alle classi PCML. L'utilizzo del PCML serializzato migliora le prestazioni del tempo di esecuzione poiché il PCML è stato già analizzato. Se si sceglie di serializzare il sorgente PCML, è necessario fare ciò quando si crea l'applicazione.

Le classi PCML interagiscono anche con le classi di IBM Toolbox per Java, che in questo esempio utilizzano il server di chiamata al programma distribuito iSeries per richiamare informazioni dal server.

Queste azioni ed interazioni abilitano il passaggio delle informazioni tra l'applicazione Java e i programmi iSeries.

Sintassi PCML

PCML è costituito dalle seguenti tag, ognuna delle quali ha tag di attributo proprie:

- La tag program si trova all'inizio e alla fine del codice che descrive il programma
- La tag struct definisce una struttura con nome che può essere specificata come argomento in un programma o come campo in un'altra struttura con nome.Una tag struct contiene una tag data o struct per ogni campo nella struttura.
- La tag data definisce un campo all'interno di un programma o di una struttura.

Nell'esempio seguente la sintassi PCML descrive un programma con una categoria di dati e alcuni dati isolati.

```
<program>

    <struct>
        <data> </data>
    </struct>
```

```
    <data> </data>

</program>
```

Tag del programma PCML:

La tag del programma PCML può essere estesa mediante i seguenti elementi:

```
<program name="name"
    [ entrypoint="entry-point-name" ]
    [ epccsid="ccsid" ]
    [ path="path-name" ]
    [ parseorder="name-list" ]
    [ returnvalue="{ void | integer }" ]
    [ threadsafe="{ true | false }" ]>
</program>
```

La seguente tabella elenca gli attributi della tag del programma. Ogni voce include il nome dell'attributo, i possibili valori validi e una descrizione dell'attributo

Attributo	Valore	Descrizione
entrypoint=	*entry-point-name*	Specifica il nome del punto d'immissione in un oggetto del programma di servizio che rappresenta la destinazione di questa chiamata al programma.
epccsid=	*ccsid*	Indica il CCSID del punto d'immissione in un programma di servizio. Per ulteriori informazioni, consultare le note relative all'immissione del programma di servizio in ServiceProgramCall javadoc.
name=	*name*	Specifica il nome del programma.

Attributo	Valore	Descrizione
path=	*path-name*	Specifica il percorso per l'oggetto del programma.Il valore predefinito consiste nel presupporre che il programma si trovi nella libreria QSYS. Il percorso deve essere un nome percorso IFS valido per un oggetto *PGM o *SRVPGM. Se viene richiamato un oggetto *SRVPGM, è necessario specificare l'attributo entrypoint per indicare il nome del punto d'immissione da richiamare. Se non viene specificato l'attributo entrypoint, il valore predefinito per tale attributo si presuppone che sia un oggetto *PGM dalla libreria QSYS. Se viene specificato l'attributo entrypoint, il valore predefinito per tale attributo si presuppone che sia un oggetto *SRVPGM nella libreria QSYS. Il nome percorso deve essere specificato con tutti i caratteri maiuscoli. No utilizzare l'attributo **path** quando l'applicazione deve impostare il percorso in fase di esecuzione, ad esempio, quando un utente specifica quale libreria viene utilizzata per l'installazione. In questo caso, utilizzare il metodo ProgramCallDocument.setPath().

Attributo	Valore	Descrizione
parseorder=	*name-list*	Specifica l'ordine in cui verranno elaborati i parametri di emissione. Il valore specificato è un elenco di nomi parametro separata da spazi parametro in cui devono essere elaborati i parametri. I nomi nell'elenco devono essere uguali ai nomi specificati nell'attributo **name** delle tag appartenenti a **<program>**. Il valore predefinito consiste nell'elaborare i parametri d'emissione nell'ordine in cui le tag appaiono nel documento. Alcuni programmi restituiscono informazioni in un parametro che descrivono le informazioni contenute in un parametro precedente. Ad esempio, supponiamo che un programma restituisca una schiera di strutture nel primo parametro e il numero di voci nella schiera nel secondo parametro. In questo caso, è necessario elaborare il secondo parametro affinché il programma ProgramCallDocument determini il numero di strutture da elaborare nel primo parametro.
returnvalue=	*void* Il programma non restituisce un valore. *integer* Il programma restituisce un valore intero con segno a 4 byte.	Specifica, se esiste, il tipo di valore restituito da una chiamata al programma di servizio. Tale attributo non è valido per le chiamate all'oggetto *PGM.
threadsafe=	*true* Il programma viene considerato protetto per il sottoprocesso. *false* Il programma non è protetto per il sottoprocesso.	Quando si richiamano un programma Java ed un programma iSeries che si trovano sullo stesso server, utilizzare questa proprietà per specificare se si desidera richiamare il programma iSeries nello stesso lavoro e sullo stesso sottoprocesso del programma Java. Se è noto che il programma è protetto per il sottoprocesso, l'impostazione della funzione su *true* produrrà prestazioni migliori. Per mantenere l'ambiente protetto, il valore predefinito consiste nel richiamare programmi in lavori server separati. Il valore predefinito è *false*.

Tag struct PCML:

La tag struct PCML può essere estesa mediante i seguenti elementi:

```
<struct name="name"
    [ count="{number | data-name }"]
    [ maxvrm="version-string" ]
```

```
    [ minvrm="version-string" ]
    [ offset="{number | data-name }" ]
    [ offsetfrom="{number | data-name | struct-name }" ]
    [ outputsize="{number | data-name }" ]
    [ usage="{ inherit | input | output | inputoutput }" ]>
</struct>
```

La seguente tabella elenca gli attributi della tag struct. Ogni voce include il nome dell'attributo, i possibili valori validi e una descrizione dell'attributo

Attributo	Valore	Descrizione
name=	*name*	Specifica il nome dell'elemento **<struct>**
count=	*number* dove *number* definisce una schiera dimensionata fissa, non modificabile. *data-name* dove *data-name* definisce il nome di un elemento **<data>** all'interno di un documento PCML che conterrà, in fase di esecuzione, il numero degli elementi nella schiera. E' possibile che *data-name* specificato sia un nome completo o un nome relativo all'elemento corrente. In entrambi i casi, è necessario che il nome faccia riferimento all'elemento **<data>** definito con **type=***"int"*. Consultare Definizione dei nomi relativi per ulteriori informazioni sulle modalità di definizione dei nomi relativi.	Specifica se l'elemento è una schiera e identifica il numero di voci nella schiera. Se quest'attributo viene omesso, l'elemento non è definito come una schiera, anche se è possibile sia contenuto all'interno di un altro elemento definito come una schiera.
maxvrm=	*version-string*	Specifica la versione più vecchia di OS/400 sulla quale esiste questo elemento. Se la versione di OS/400 è maggiore della versione specificata su questo attributo, questo elemento ed i suoi secondari, se ne esistono, non vengono elaborati durante una chiamata al programma. L'elemento **maxvrm** è utile per definire le interfacce di programma che differiscono tra i release di OS/400. La sintassi della stringa di versione deve essere ″VvRrMm,″ dove le lettere maiuscole ″V,″ ″R,″ e ″M″ sono caratteri costanti letterali e ″v,″ ″r″ e ″m″ indicano una o più cifre rappresentanti la versione, il release e il livello di modifica. Il valore per ″v″ deve essere incluso tra 1 e 255. Il valore per ″r″ e ″m″ deve essere incluso tra 0 e 255.

Attributo	Valore	Descrizione
minvrm=	*version-string*	Specifica la versione più recente di OS/400 sulla quale esiste questo elemento. Se la versione OS/400 è più recente della versione specificate su questo attributo, questo elemento e i suoi secondari, se ne esistono, non vengono elaborati durante una chiamata al programma. Questo attributo è utile per definire le interfacce di programma che differiscono tra i release di OS/400. La sintassi della stringa di versione deve essere ″VvRrMm,″ dove le lettere maiuscole ″V,″ ″R,″ e ″M″ sono caratteri costanti letterali e ″v,″ ″r″ e ″m″ indicano una o più cifre rappresentanti la versione, il release e il livello di modifica. Il valore per ″v″ deve essere incluso tra 1 e 255. Il valore per ″r″ e ″m″ deve essere incluso tra 0 e 255.

Attributo	Valore	Descrizione
offset=	*number* dove *number* definisce uno scostamento fisso, non modificabile. *data-name* dove *data-name* definisce il nome di un elemento **<data>** in un documento PCML che conterrà, in fase di esecuzione, lo scostamento per tale elemento. E' possibile che **data-name** specificato sia un nome completo o un nome relativo all'elemento corrente. In entrambi i casi, è necessario che il nome faccia riferimento all'elemento **<data>** definito con **type=***"int"*. Consultare Definizione dei nomi relativi per ulteriori informazioni sulle modalità di definizione dei nomi relativi.	Indica lo scostamento per l'elemento **<struct>** in un parametro di emissione. Alcuni programmi restituiscono informazioni con una struttura fissa seguite da uno o più campi o strutture a lunghezza variabile. I questo caso, l'ubicazione di un elemento a lunghezza variabile viene solitamente indicata come uno scostamento o spostamento in un parametro. L'attributo **offset** viene utilizzato per descrivere lo scostamento per l'elemento **<struct>**. **Offset** viene utilizzato insieme all'attributo **offsetfrom**.Se l'attributo **offsetfrom** non viene specificato, l'ubicazione di base per lo scostamento specificato nell'attributo **offset** è il principale dell'elemento. Consultare Specifica scostamenti per ulteriori informazioni su come utilizzare gli attributi **offset** e **offsetfrom** . Gli attributi **offset** e **offsetfrom** vengono utilizzati solo per elaborare i dati di emissione da un programma. Questi attributi non controllano lo scostamento o lo spostamento dei dati di immissione. Se questo attributo viene omesso, l'ubicazione dei dati relativi a questo elemento viene immediatamente dopo il precedente elemento nel parametro, se presente.

Attributo	Valore	Descrizione
offsetfrom=	*number* dove *number* definisce un'ubicazione di base fissa, non modificabile. Un attributo *number* è più tipicamente utilizzato per specificare **number=***"0"* che indica che lo scostamento è uno scostamento assoluto dall'inizio del parametro. *data-name* dove *data-name* definisce il nome di un elemento **<data>** da utilizzare come ubicazione di base per lo scostamento. Il nome dell'elemento specificato deve essere il principale o l'origine di questo elemento. Il valore dall'attributo **offset** sarà relativo all'ubicazione dell'elemento specificato su questo attributo.E' possibile che *data-name* specificato sia un nome completo o un nome relativo all'elemento corrente. In entrambi i casi, il nome deve riferirsi ad un'origine di questo elemento. Consultare Definizione dei nomi relativi per ulteriori informazioni sulle modalità di definizione dei nomi relativi. *struct-name* dove *struct-name* definisce il nome di un elemento **<struct>** da utilizzare come ubicazione di base per lo scostamento. Il nome dell'elemento specificato deve essere il principale o l'origine di questo elemento. Il valore dall'attributo **offset** sarà relativo all'ubicazione dell'elemento specificato su questo attributo.Lo *struct-name* specificato deve essere un nome completo o un nome relativo all'elemento corrente. In entrambi i casi, il nome deve riferirsi ad un'origine di questo elemento. Consultare Definizione dei nomi relativi per ulteriori informazioni sulle modalità di definizione dei nomi relativi.	Specifica l'ubicazione di base alla quale si riferisce l'attributo **offset**. Se l'attributo **offsetfrom** non è specificato, l'ubicazione di base per lo scostamento specificato sull'attributo**offset** è l'origine dell'elemento. Consultare Specifica scostamenti per ulteriori informazioni su come utilizzare gli attributi **offset** e **offsetfrom** . Gli attributi **offset** e **offsetfrom** vengono utilizzati solo per elaborare i dati di emissione da un programma. Questi attributi non controllano lo scostamento o lo spostamento dei dati di immissione.

Attributo	Valore	Descrizione
outputsize=	*number* dove *number* definisce a un numero fisso, non modificabile di byte da conservare. *data-name* dove *data-name* definisce il nome di un elemento **<data>** in un documento PCML che conterrà, in fase di esecuzione, il numero di byte da conservare per i dati di emissione. E' possibile che *data-name* specificato sia un nome completo o un nome relativo all'elemento corrente. In entrambi i casi, è necessario che il nome faccia riferimento all'elemento **<data>** definito con **type=***"int"*. Consultare Definizione dei nomi relativi per ulteriori informazioni sulle modalità di definizione dei nomi relativi.	Specifica il numero di byte da riservare per i dati di emissione per l'elemento. Per i parametri di emissione con una lunghezza variabile, è necessario l'attributo **outputsize** per specificare il numero di byte che è necessario conservare per i dati che devono essere restituiti dal programma server. **Outputsize** può essere specificato su tutti i campi a lunghezza variabile e le schiere a dimensione variabile o può essere specificato per un parametro intero che contiene uno o più campi a lunghezza variabile. **Outputsize** non è necessario e non deve essere specificato per i parametro di emissione a dimensione fissa. Il valore specificato sull'attributo è utilizzato come dimensione totale per l'elemento che include tutti i secondari dell'elemento. Pertanto, l'attributo **outputsize** viene ignorato su ogni secondario o discendente dell'elemento. Se l'attributo viene omesso, il numero di byte da conservare per i dati di emissione viene stabilito in fase di esecuzione mediante l'aggiunta del numero di byte da conservare per tutti i valori secondari dell'elemento **<struct>**.
usage=	*inherit*	L'utilizzo viene ereditato dall'elemento principale. Se la struttura non ha una struttura principale, si presume che l'utilizzo sia**inputoutput**.
	input	La struttura rappresenta un valore di immissione al programma host. Per i tipi numerici e di carattere, viene eseguita l'appropriata conversione.
	output	La struttura rappresenta un valore di emissione dal programma host. Per i tipi numerici e di carattere, viene eseguita l'appropriata conversione.
	inputoutput	La struttura rappresenta sia un valore di immissione che di emissione.

Specifica degli scostamenti

Alcuni programmi restituiscono informazioni con una struttura fissa seguite da uno o più campi o strutture a lunghezza variabile. I questo caso, l'ubicazione di un elemento a lunghezza variabile viene solitamente indicata come uno scostamento o spostamento in un parametro.

Uno scostamento è la distanza in byte dall'inizio dei parametri all'inizio di un campo o di una struttura. Uno spostamento è la distanza in byte dall'inizio di una struttura all'inizio di un'altra struttura.

Per gli scostamenti, poiché la distanza viene considerata dall'inizio del parametro, è necessario specificare **offsetfrom="0"**. Il seguente è un esempio di uno scostamento dall'inizio del parametro:

```
<pcml version="1.0">
  <program name="myprog" path="/QSYS.lib/MYLIB.lib/MYPROG.pgm">
    <!-- la variabile del programma di ricezione contiene il percorso -->
    <struct name="receiver" usage="output" outputsize="2048">
      <data name="pathType"         type="int"  length="4" />
      <data name="offsetToPathName" type="int"  length="4" />
      <data name="lengthOfPathName" type="int"  length="4" />
      <data name="pathName"         type="char" length="lengthOfPathName"
              offset="offsetToPathName"  offsetfrom="0"/>
    </struct>
  </program>
</pcml>
```

Per gli spostamenti, dal momento che la distanza è considerata dall'inizio di un'altra struttura, si specifica il nome della struttura al quale si riferisce lo scostamento. Il seguente è un esempio di uno spostamento dall'inizio di una struttura con nome:

```
<pcml ="1.0">
  <program name="myprog" path="/QSYS.lib/MYLIB.lib/MYPROG.pgm">
    <!-- la variabile del programma di ricezione contiene un oggetto -->
    <struct name="receiver" usage="output" >
      <data name="objectName"       type="char"  length="10" />
      <data name="libraryName"      type="char"  length="10" />
      <data name="objectType"       type="char"  length="10" />
      <struct name="pathInfo" usage="output" outputsize="2048" >
        <data name="pathType"         type="int"  length="4" />
        <data name="offsetToPathName" type="int"  length="4" />
        <data name="lengthOfPathName" type="int"  length="4" />
        <data name="pathName"         type="char" length="lengthOfPathName"
                offset="offsetToPathName"  offsetfrom="pathInfo"/>
      </struct>
    </struct>
  </program>
</pcml>
```

Tag Data PCML:

La tag Data PCML può avere i seguenti attributi. Gli attributi racchiusi tra parentesi, [], indicano che l'attributo è facoltativo. Se si specifica un attributo facoltativo, non includere le parentesi nel proprio sorgente. Alcuni valori di attributo vengono visualizzati come un elenco di opzioni racchiuse tra parentesi graffe, {}, con le scelte possibili separate da barre verticali, |. Quando si specifica uno di questi attributi, non includere le parentesi graffe nel sorgente e specificare solo una delle scelte visualizzate.

```
<data type="{ char | int | packed | zoned | float | byte | struct }"
    [ bidistringtype="{ ST4 | ST5 | ST6 | ST7 | ST8 | ST9 | ST10 | ST11 | DEFAULT }"]
    [ ccsid="{ number | data-name }" ]
    [ chartype="{ onebyte | twobyte }"]
    [ count="{ number | data-name }" ]
    [ init="string" ]
    [ length="{ number | data-name }" ]
    [ maxvrm="version-string" ]
    [ minvrm="version-string" ]
    [ name="name" ]
```

```
    [ offset="{ number | data-name }" ]
    [ offsetfrom="{ number | data-name | struct-name }" ]
    [ outputsize="{ number | data-name | struct-name }" ]
    [ passby= "{ reference | value }" ]
    [ precision="number" ]
    [ struct="struct-name" ]
    [ trim="{ right | left | both | none }" ]
    [ usage="{ inherit | input | output | inputoutput }" ]>
</data>
```

La seguente tabella elenca gli attributi della tag Data. Ogni voce include il nome dell'attributo, i possibili valori validi e una descrizione dell'attributo

Attributo	Valore	Descrizione
type=	*char* dove *char* indica un valore di carattere. Un valore dati *char* viene restituito come un *java.lang.String*.Per ulteriori informazioni, consultare i *valori char* per length. *int* dove *int* indica un valore di numero intero. Un valore dati *int* viene restituito come un *java.lang.Long*. Per ulteriori informazioni, consultare i valori *int* per length e precision. *packed* dove *packed* indica un valore decimale compresso. Un valore dati *packed* viene restituito come un *java.math.BigDecimal*. Per ulteriori informazioni, consultare i *valori packed* per length e precision. *zoned* dove *zoned* indica un valore decimale a zonatura. Un valore dati *zoned* viene restituito come un *java.math.BigDecimal*. Per ulteriori informazioni, consultare i *valori zoned* per length e precision. *float* dove *float* indica un valore virgola mobile. L'attributo **length** specifica il numero di byte, ″4″ o ″8″. Un numero intero a 4-byte è restituito come *java.lang.Float*. Un numero intero a 8 byte viene restituito come un *java.lang.Double*. Per ulteriori informazioni, consultare i *valori float* per length. *byte* dove *byte* indica un valore espresso in byte. Non viene eseguita alcuna conversione sui dati. Un valore dati *byte* viene restituito come una schiera di valori *byte* (*byte[]*). Per ulteriori informazioni, consultare i *valori byte* per length. *struct* dove *struct* indica il nome dell'elemento **<struct>**. Un elemento *struct* consente di definire una struttura una volta e di riutilizzarla più volte all'interno del documento.Quando si utilizza **type=***″struct″*, è come se la struttura specificata venisse visualizzata in questa ubicazione nel documento.Un elemento *struct* non prevede un valore length e non ha un valore per precision.	Indica il tipo di dati che si sta utilizzando (char, integer, packed, zoned, floating point, byte o struct). I valori degli attributi length e precision sono differenti per diversi tipi di dati. Per ulteriori informazioni, consultare i valori per length e precision.

Attributo	Valore	Descrizione
bidistringtype=	*DEFAULT* dove DEFAULT è il tipo stringa predefinito per i dati non bidirezionali (LTR). *ST4* dove *ST4* è Tipo stringa 4. *ST5* dove *ST5* è Tipo stringa 5. *ST6* dove *ST6* è Tipo stringa 6. *ST7* dove *ST7* è Tipo stringa 7. *ST8* dove *ST8* è Tipo stringa 8. *ST9* dove *ST9* è Tipo stringa 9. *ST10* dove *ST10* è Tipo stringa 10. *ST11* dove *ST11* è Tipo stringa 11.	Indica il tipo di stringa bidirezionale per gli elementi **<data>** con **type=***"char"*. Se questo attributo viene omesso, il tipo di stringa per questo elemento è implicato dal CCSID (se esplicitamente specificato il CCSID predefinito dell'ambiente host). I tipi di stringa sono definiti nella Javadoc per la classe BidiStringType.
ccsid=	*number* dove *number* definisce un CCSID fisso, non modificabile. *data-name* dove *data-name* definisce il nome che conterrà, in fase di esecuzione, il CCSID dei dati carattere. E' possibile che *data-name* specificato sia un nome completo o un nome relativo all'elemento corrente. In entrambi i casi, è necessario che il nome faccia riferimento all'elemento **<data>** definito con **type=***"int"*. Consultare Definizione dei nomi relativi per ulteriori informazioni sulle modalità di definizione dei nomi relativi.	Indica il CCSID dell'host per i dati carattere per l'elemento **<data>**. L'attributo **ccsid** può essere specificato solo per gli elementi **<data>** con **type=***"char"*. Se questo attributo viene omesso, si presume che i dati carattere per questo elemento siano nel CCSID predefinito dell'ambiente host.
chartype=	*onebyte* dove *onebyte* indica la dimensione di ciascun carattere. *twobyte* dove *twobyte* indica la dimensione di ciascun carattere. Quando si utilizza *chartype*, l'attributo **length=***"number"* indica il numero di caratteri e non il numero di byte.	Indica la dimensione di ciascun carattere.

Attributo	Valore	Descrizione
count=	*number* dove *number* definisce un numero fisso, non modificabile di elementi in una schiera dimensionata. *data-name* dove *data-name* definisce il nome di un elemento **<data>** all'interno di un documento PCML che conterrà, in fase di esecuzione, il numero degli elementi nella schiera. E' possibile che *data-name* specificato sia un nome completo o un nome relativo all'elemento corrente. In entrambi i casi, è necessario che il nome faccia riferimento all'elemento **<data>** definito con **type=***"int"*. Consultare Definizione dei nomi relativi per ulteriori informazioni sulle modalità di definizione dei nomi relativi.	Specifica se l'elemento è una schiera e identifica il numero di voci nella schiera. Se l'attributo *count* viene omesso, l'elemento non è definito come una schiera, anche se è possibile che sia contenuto all'interno di un altro elemento definito come una schiera.
init=	*string*	Indica un valore iniziale per l'elemento **<data>**. Il valore *init* viene utilizzato se un valore iniziale non è impostato esplicitamente dal programma di applicazione quando vengono utilizzati gli elementi **<data>** con **usage=***"input"* o **usage=***"inputoutput"*. Si utilizza il valore iniziale specificato per inizializzare i valori scalari. Se l'elemento è definito come una schiera o è contenuto all'interno di una struttura definita come una schiera, il valore iniziale specificato viene utilizzato come un valore iniziale per tutte le voci nella schiera.

Attributo	Valore	Descrizione
length=	*number* dove *number* definisce il numero di byte richiesti dai dati. Tuttavia, quando si utilizza l'attributo *chartype*, *number* indica il numero di caratteri e non il numero di byte. *data-name* dove *data-name* definisce il nome di un elemento **<data>** all'interno di un documento PCML che conterrà, in fase di esecuzione, la lunghezza. E' possibile specificare *data-name* solo per gli elementi **<data>** con **type=***"char"* o **type=***"byte"*. E' possibile che *data-name* specificato sia un nome completo o un nome relativo all'elemento corrente. In entrambi i casi, è necessario che il nome faccia riferimento all'elemento **<data>** definito con **type=***"int"*. Consultare Definizione dei nomi relativi per ulteriori informazioni sulle modalità di definizione dei nomi relativi.	Indica la lunghezza dell'elemento data. La variazione nell'utilizzo di questo attributo dipende dal tipo di dati. Per ulteriori informazioni, consultare i valori per length e precision.
maxvrm=	*version-string*	Specifica la versione meno recente di iSeries sulla quale esiste questo elemento. Se la versione iSeries è successiva alla versione specificata su questo attributo, questo elemento ed i suoi secondari, se ne esistono, non vengono elaborati durante una chiamata ad un programma. Questo attributo è utile per definire le interfacce di programma che differiscono tra i release di iSeries. La sintassi della stringa di versione deve essere ″VvRrMm″, dove le lettere maiuscole ″V,″ ″R,″ e ″M″ sono caratteri costanti letterali e ″v,″ ″r,″ e ″m″ sono una o più cifre rappresentanti la versione, il release e il livello di modifica. Il valore per ″v″ deve essere incluso tra 1 e 255. Il valore per ″r″ e ″m″ deve essere incluso tra 0 e 255.

Attributo	Valore	Descrizione
minvrm=	*version-string*	Specifica la versione più recente di iSeries sulla quale esiste questo elemento. Se la versione iSeries è più recente della versione specificata su questo attributo, questo elemento e i suoi secondari, se ne esistono, non vengono elaborati durante una chiamata ad un programma. Questo attributo è utile per definire le interfacce di programma che differiscono tra i release di iSeries. La sintassi della stringa di versione deve essere ″VvRrMm,″ dove le lettere maiuscole ″V,″ ″R,″ e ″M″ sono caratteri costanti letterali e ″v,″ ″r″ e ″m″ indicano una o più cifre rappresentanti la versione, il release e il livello di modifica. Il valore per″v″ deve essere incluso tra 1 e 255. Il valore per ″r″ e ″m″ deve essere incluso tra 0 e 255.
name=	*name*	Indica il nome dell'elemento **<data>**.
offset=	*number* dove *number* definisce uno scostamento fisso, non modificabile. *data-name* dove *data-name* definisce il nome di un elemento **<data>** in un documento PCML che conterrà, in fase di esecuzione, lo scostamento a questo elemento. E' possibile che *data-name* specificato sia un nome completo o un nome relativo all'elemento corrente. In entrambi i casi, è necessario che il nome faccia riferimento all'elemento **<data>** definito con **type=***″int″*. Consultare Definizione dei nomi relativi per ulteriori informazioni sulle modalità di definizione dei nomi relativi.	Indica lo scostamento all'elemento **<data>** in un parametro di emissione. Alcuni programmi restituiscono informazioni con una struttura fissa seguite da uno o più campi o strutture a lunghezza variabile. I questo caso, l'ubicazione di un elemento a lunghezza variabile viene solitamente indicata come uno scostamento o spostamento in un parametro. Un attributo **offset** viene utilizzato in congiunzione con l'attributo **offsetfrom**. Se l'attributo **offsetfrom** non è specificato, l'ubicazione di base per lo scostamento specificato sull'attributo**offset** è l'origine dell'elemento. Consultare Specifica scostamenti per ulteriori informazioni su come utilizzare gli attributi **offset** e **offsetfrom** . Gli attributi **offset** e **offsetfrom** vengono utilizzati solo per elaborare i dati di emissione da un programma. Questi attributi non controllano lo scostamento o lo spostamento dei dati di immissione. Se questo attributo viene omesso, l'ubicazione dei dati relativi a questo elemento viene immediatamente dopo il precedente elemento nel parametro, se presente.

Attributo	Valore	Descrizione
offsetfrom=	*number* dove *number* definisce un'ubicazione di base fissa, non modificabile. *Number* è più tipicamente utilizzato per specificare **number=***"0"* che indica che lo scostamento è uno scostamento assoluto dall'inizio del parametro. *data-name* dove *data-name* definisce il nome di un elemento **<data>** utilizzato come ubicazione di base per lo scostamento. Il nome dell'elemento specificato deve essere il principale o l'origine di questo elemento. Il valore dall'attributo **offset** sarà relativo all'ubicazione dell'elemento specificato su questo attributo. E' possibile che *data-name* specificato sia un nome completo o un nome relativo all'elemento corrente. In entrambi i casi, il nome deve riferirsi ad un'origine di questo elemento. Consultare Definizione dei nomi relativi per ulteriori informazioni sulle modalità di definizione dei nomi relativi. *struct-name* dove *struct-name* definisce il nome di un elemento **<struct>** utilizzato come ubicazione di base per lo scostamento. Il nome dell'elemento specificato deve essere il principale o l'origine di questo elemento. Il valore dall'attributo **offset** sarà relativo all'ubicazione dell'elemento specificato su questo attributo.Lo *struct-name* specificato deve essere un nome completo o un nome relativo all'elemento corrente. In entrambi i casi, il nome deve riferirsi ad un'origine di questo elemento. Consultare Definizione dei nomi relativi per ulteriori informazioni sulle modalità di definizione dei nomi relativi.	Specifica l'ubicazione di base alla quale si riferisce l'attributo **offset**. Se l'attributo **offsetfrom** non è specificato, l'ubicazione di base per lo scostamento specificato sull'attributo**offset** è l'origine dell'elemento. Consultare Specifica scostamenti per ulteriori informazioni su come utilizzare gli attributi **offset** e **offsetfrom** . Gli attributi **offset** e **offsetfrom** vengono utilizzati solo per elaborare i dati di emissione da un programma. Questi attributi non controllano lo scostamento o lo spostamento dei dati di immissione.

Attributo	Valore	Descrizione
outputsize=	*number* dove *number* definisce un numero fisso, non modificabile di byte da conservare. *data-name* dove *data-name* definisce il nome di un elemento **<data>** in un documento PCML che conterrà, in fase di esecuzione, il numero di byte da conservare per i dati di emissione. E' possibile che *data-name* specificato sia un nome completo o un nome relativo all'elemento corrente. In entrambi i casi, è necessario che il nome faccia riferimento all'elemento **<data>** definito con **type=***"int"*. Consultare Definizione dei nomi relativi per ulteriori informazioni sulle modalità di definizione dei nomi relativi.	Specifica il numero di byte da riservare per i dati di emissione per l'elemento. Per i parametri di emissione con lunghezza variabile, è necessario l'attributo **outputsize** per specificare la quantità di byte che deve essere conservata per fare in modo che i dati vengano restituiti dal programma iSeries. Un attributo **outputsize** può essere specificato su tutti i campi a lunghezza variabile e le schiere a dimensione variabile o può essere specificato per un parametro intero che contiene uno o più campi a lunghezza variabile. **Outputsize** non è necessario e non deve essere specificato per i parametro di emissione a dimensione fissa. Il valore specificato su questo attributo è utilizzato come dimensione totale per l'elemento che include tutti i secondari dell'elemento. Perciò, l'attributo **outputsize** viene ignorato su ogni secondario o discendente dell'elemento. Se **outputsize** viene omesso, il numero di byte da conservare per i dati di emissione viene stabilito in fase di esecuzione mediante l'aggiunta di un numero di byte da conservare per tutti i valori secondari dell'elemento **<struct>**.
passby=	*reference* dove *reference* indica che il parametro verrà inoltrato per riferimento. Quando il programma viene richiamato, esso passerà un puntatore al valore di parametro. *value* dove *value* indica un valore di numero intero. Tale valore è consentito solo quando viene specificato **type=** *"int"* e **length=***"4"*.	Indica se il parametro viene passato per riferimento o per valore. Questo attributo è consentito solo quando questo elemento è un valore secondario di un elemento **<program>** che definisce una chiamata al programma di servizio.
precision=	*number*	Specifica il numero di byte di precisione per alcuni tipi di dati numerici. Per ulteriori informazioni, consultare i valori per length e precision.
struct=	*name*	Indica il nome di un elemento **<struct>** per l'elemento **<data>**. E' possibile specificare un attributo **struct** solo per gli elementi **<data>** con **type=***"struct"*.

Attributo	Valore	Descrizione
trim=	*right* dove *right* è il funzionamento predefinito utilizzato per ridimensionare gli spazi bianchi finali. *left* dove *left* viene utilizzato per ridimensionare gli spazi bianchi iniziali. *both* dove *both* viene utilizzato per ridimensionare gli spazi bianchi iniziali e finali. *none* dove *none* indica che gli spazi bianchi non vengono ridimensionati.	Indica come ridimensionare gli spazi bianchi dai dati carattere.
usage=	*inherit*	L'utilizzo viene ereditato dall'elemento principale. Se la struttura non ha una struttura principale, si presume che l'utilizzo sia*inputoutput*.
	input	Definisce un valore di immissione al programma host. Per i tipi numerici e di carattere, viene eseguita l'appropriata conversione.
	output	Definisce un valore di emissione dal programma host. Per i tipi numerici e di carattere, viene eseguita l'appropriata conversione.
	inputoutput	Definisce sia i valori di immissione che di emissione.

Specifica degli scostamenti

Alcuni programmi restituiscono informazioni con una struttura fissa seguite da uno o più campi o strutture a lunghezza variabile. I questo caso, l'ubicazione di un elemento a lunghezza variabile viene solitamente indicata come uno scostamento o spostamento in un parametro.

Uno scostamento è la distanza in byte dall'inizio dei parametri all'inizio di un campo o di una struttura. Uno spostamento è la distanza in byte dall'inizio di una struttura all'inizio di un'altra struttura.

Per gli scostamenti, poiché la distanza viene considerata dall'inizio del parametro, è necessario specificare **offsetfrom=**″**0**″.Il seguente è un esempio di uno scostamento dall'inizio del parametro:

```
<pcml version="1.0">
  <program name="myprog" path="/QSYS.lib/MYLIB.lib/MYPROG.pgm">
    <!-- la variabile del programma di ricezione contiene il percorso -->
    <struct name="receiver" usage="output" outputsize="2048">
      <data name="pathType"         type="int"  length="4" />
      <data name="offsetToPathName" type="int"  length="4" />
      <data name="lengthOfPathName" type="int"  length="4" />
      <data name="pathName"         type="char" length="lengthOfPathName"
                offset="offsetToPathName"  offsetfrom="0"/>
    </struct>
  </program>
</pcml>
```

Per gli spostamenti, dal momento che la distanza è considerata dall'inizio di un'altra struttura, si specifica il nome della struttura al quale si riferisce lo scostamento. Il seguente è un esempio di uno spostamento dall'inizio di una struttura con nome:

```
<pcml version="1.0">
  <program name="myprog" path="/QSYS.lib/MYLIB.lib/MYPROG.pgm">
    <!-- la variabile del programma di ricezione contiene un oggetto -->
    <struct name="receiver" usage="output" >
      <data name="objectName"       type="char"  length="10" />
      <data name="libraryName"      type="char"  length="10" />
      <data name="objectType"       type="char"  length="10" />
      <struct name="pathInfo" usage="output" outputsize="2048" >
        <data name="pathType"         type="int"  length="4" />
        <data name="offsetToPathName" type="int"  length="4" />
        <data name="lengthOfPathName" type="int"  length="4" />
        <data name="pathName"         type="char" length="lengthOfPathName"
                  offset="offsetToPathName"  offsetfrom="pathInfo"/>
      </struct>
    </struct>
  </program>
</pcml>
```

Valori per length e precision:

I valori degli attributi length e precision sono differenti per diversi tipi di dati. La seguente tabella elenca ogni tipo di dati con una descrizione dei possibili valori length e precision.

Tipo dati	Length	Precision
type="char"	Il numero di byte di dati per questo elemento, che non è necessariamente il numero dei caratteri. E' necessario specificare un numero costante letterale *number* costante letterale o un *data-name*.	Non applicabile
type="int"	Il numero di byte di dati per questo elemento: 2, 4 o 8. E' necessario specificare un *number* costante letterale.	Indica il numero di bit di precisione e se il numero intero è con segno o senza segno: • Per **length**="2" – Utilizzare **precision**="15" per un numero intero a 2 byte con segno. Questo è un valore predefinito – Utilizzare **precision**="16" per un numero intero a 2 byte senza segno • Per **length**="4" – Utilizzare **precision**="31" per un numero intero a 4 byte con segno – Utilizzare **precision**="32" per un numero intero a 4 byte senza segno • Per **length**="8" utilizzare **precision**="63" per un numero intero a 8 byte con segno

Tipo dati	Length	Precision
type="packed" o "zoned"	Il numero di cifre numeriche di dati per questo elemento. E' necessario specificare un *number* costante letterale.	Il numero di cifre decimali per l'elemento. Questo numero deve essere maggiore o uguale a zero e minore o uguale al numero totale di cifre specificate sull'attributo **length**.
type="float"	Il numero di byte, 4 o 8, di dati per questo elemento.E' necessario specificare un *number* costante letterale.	Non applicabile
type="byte"	Il numero di byte di dati per questo elemento. E' necessario specificare un *number* costante letterale o *data-name*.	Non applicabile
type="struct"	Non consentito.	Non applicabile

Definizione dei nomi relativi

Numerosi attributi consentono di specificare il nome di un altro elemento, o tag, nel documento come valore di attributo. E' possibile che il nome specificato sia un nome che è relativo all'etichetta corrente.

I nomi vengono definiti controllando se il nome può essere definito come un secondario o discendente della tag contenente la tag corrente. Se non è possibile definire il nome a questo livello, la ricerca continua con la tag di contenimento a livello più elevato. Alla fine, questa definizione deve terminare con una corrispondenza di tag contenuta nella tag **<pcml>** o nella tag <**rfml**>, in tal caso il nome viene considerato come assoluto e non come relativo.

Segue un esempio di utilizzo di PCML:

```
<pcml version="1.0">
  <program name="polytest" path="/QSYS.lib/MYLIB.lib/POLYTEST.pgm">
    <!-- Il parametro 1 contiene un conteggio dei poligoni insieme a una schiera di poligoni -->
    <struct name="parm1" usage="inputoutput">
      <data name="nbrPolygons" type="int" length="4" init="5" />
      <!-- Ciascun poligono contiene un conteggio del numero di punti insieme a una schiera di punti -->
      <struct name="polygon" count="nbrPolygons">
        <data name="nbrPoints" type="int" length="4" init="3" />
        <struct name="point" count="nbrPoints" >
          <data name="x" type="int" length="4" init="100" />
          <data name="y" type="int" length="4" init="200" />
        </struct>
      </struct>
    </struct>
  </program>
</pcml>
```

Segue un esempio di utilizzo di RFML:

```
<rfml version="4.0">
  <struct name="polygon">
    <!-- Ciascun poligono contiene un conteggio del numero di punti insieme a una schiera di punti. -->
    <data name="nbrPoints" type="int" length="4" init="3" />
    <data name="point" type="struct" struct="point" count="nbrPoints" />
  </struct>
  <struct name="point" >
    <data name="x" type="int" length="4" init="100" />
    <data name="y" type="int" length="4" init="200" />
  </struct>
  <recordformat name="polytest">
    <!-- Questo formato contiene un conteggio dei poligoni insieme a una schiera di poligoni -->
```

```
    <data name="nbrPolygons" type="int" length="4" init="5" />
    <data name="polygon" type="struct" struct="polygon" count="nbrPolygons" />
  </recordformat>
</rfml>
```

RFML (Record Format Markup Language)

L'RFML (Record Format Markup Language) è un'estensione di XML per specificare i formati record. Il componente RFML di IBM Toolbox per Java abilita le applicazioni all'utilizzo dei documenti RFML per specificare e manipolare i campi all'interno di un certo tipo di record.

I documenti RFML, detti file sorgente RFML, rappresentano un'utile sottoserie dei tipi di dati DDS (data description specification) definiti per i file fisici e logici sui server iSeries. E' possibile utilizzare i documenti RFML per gestire le informazioni che si trovano in:

- Record file
- Voci code dati
- Spazi utente
- Buffer dati arbitrari

Nota: per ulteriori informazioni sull'utilizzo di DDS per descrivere attributi di dati, consultare il Riferimento DDS.

RFML è molto simile a PCML (Program Call Markup Language), un'altra estensione XML supportata da IBM Toolbox per Java. RFML non è una sottoserie né una serie superiore di PCML, piuttosto un tipo di linguaggio di pari livello che aggiunge alcuni elementi ed attributi nuovi e ne omette altri.

PCML fornisce un'alternativa orientata su XML all'utilizzo delle classi ProgramCall e ProgramParameter. In modo simile, RFML fornisce un'alternativa di semplice gestione e utilizzo alle classi Record, RecordFormat e FieldDescription.

Per maggiori informazioni su RFML, consultare i seguenti argomenti:

Requisiti
Controllare le informazioni sui requisiti per l'utilizzo di RFML.

Esempio RFML
Verificare come l'utilizzo di RFML nell'applicazione riduce la quantità e talvolta la complessità del codice che è necessario scrivere. L'esempio include un file sorgente RFML di esempio.

Classe RecordFormatDocument
Consultare le informazioni su come utilizzare la classe RecordFormatDocument con altre classi Toolbox per Java per la lettura e la scrittura dei dati.

Documenti RFML e sintassi RFML
Acquisire conoscenze sui documenti RFML, denominati file sorgente RFML e la sintassi RFML come viene definita nella definizione tipo di dati RFML.

RFML è solo un modo per utilizzare XML con il proprio server. Per ulteriori informazioni sull'utilizzo di XML con i server iSeries, consultare le estensioni XML di IBM Toolbox per Java XML e XML (Extensible Markup Language).

Requisiti per l'utilizzo di RFML

Il componente RFML ha gli stessi requisiti JVM stazione di lavoro degli altri componenti di IBM Toolbox per Java.

Inoltre, per analizzare RFML in fase di esecuzione, è necessario che CLASSPATH per l'applicazione includa un programma di analisi XML. Il programma di analisi XML deve estendere la classe org.apache.xerces.parsers.SAXParser. Per ulteriori informazioni, consultare la seguente pagina:

"Programma di analisi XML e processore XSLT" a pagina 417

Nota: RFML ha gli stessi requisiti per il programma di analisi di PCML. Come con PCML, se si preserializza il file RFML, non è necessario includere un programma di analisi XML nel CLASSPATH dell'applicazione per eseguire l'applicazione stessa.

Esempio: utilizzo di RFML rispetto all'utilizzo delle classi Record di IBM Toolbox per Java

Questo esempio illustra le differenze tra l'utilizzo di RFML e l'utilizzo delle classi Record di IBM Toolbox per Java.

Utilizzando le classi Record tradizionali, si integrano le specifiche del formato dati con la logica aziendale dell'applicazione. Aggiungere, modificare o cancellare un campo significa che si deve modificare e ricompilare il codice Java. Tuttavia, utilizzando RFML si isolano le specifiche del formato dati nei file sorgente RFML che sono interamente separati dalla logica aziendale. Adattare le modifiche del campo significa modificare il file RFML, spesso senza dover modificare o ricompilare l'applicazione Java.

L'esempio presuppone che la propria applicazione tratti di record del cliente, definiti in un file sorgente RFML e denominati qcustcdt.rfml.Il file sorgente rappresenta i campi che compongono ogni record del cliente.

L'elenco riportato di seguito mostra come un'applicazione Java potrebbe interpretare un record del cliente utilizzando le classi Record, RecordFormat e FieldDescription di IBM Toolbox per Java:

```
// Il buffer contiene la rappresentazione binaria di un record di informazioni.
        byte[] bytes;

        // ... Leggere i dati record nel buffer ...

        // Impostare un oggetto RecordFormat per rappresentare un record cliente.
        RecordFormat recFmt1 = new RecordFormat("cusrec");
        recFmt1.addFieldDescription(new ZonedDecimalFieldDescription(new AS400ZonedDecimal(6, 0), "cusnum"));
        recFmt1.addFieldDescription(new CharacterFieldDescription(new AS400Text(8, 37), "lstnam"));
        recFmt1.addFieldDescription(new CharacterFieldDescription(new AS400Text(3, 37), "init"));
        recFmt1.addFieldDescription(new CharacterFieldDescription(new AS400Text(13, 37), "street"));
        recFmt1.addFieldDescription(new CharacterFieldDescription(new AS400Text(6, 37), "city"));
        recFmt1.addFieldDescription(new CharacterFieldDescription(new AS400Text(2, 37), "state"));
        recFmt1.addFieldDescription(new ZonedDecimalFieldDescription(new AS400ZonedDecimal(5, 0), "zipcod"));
        recFmt1.addFieldDescription(new ZonedDecimalFieldDescription(new AS400ZonedDecimal(4, 0), "cdtlmt"));
        recFmt1.addFieldDescription(new ZonedDecimalFieldDescription(new AS400ZonedDecimal(1, 0), "chgcod"));
        recFmt1.addFieldDescription(new ZonedDecimalFieldDescription(new AS400ZonedDecimal(6, 2), "baldue"));
        recFmt1.addFieldDescription(new ZonedDecimalFieldDescription(new AS400ZonedDecimal(6, 2), "cdtdue"));

        // Leggere il buffer di byte nell'oggetto RecordFormatDocument.
        Record rec1 = new Record(recFmt1, bytes);

        // Richiamare i valori campo.
        System.out.println("cusnum: " + rec1.getField("cusnum"));
        System.out.println("lstnam: " + rec1.getField("lstnam"));
        System.out.println("init:   " + rec1.getField("init"));
        System.out.println("street: " + rec1.getField("street"));
        System.out.println("city:   " + rec1.getField("city"));
        System.out.println("state:  " + rec1.getField("state"));
        System.out.println("zipcod: " + rec1.getField("zipcod"));
        System.out.println("cdtlmt: " + rec1.getField("cdtlmt"));
        System.out.println("chgcod: " + rec1.getField("chgcod"));
        System.out.println("baldue: " + rec1.getField("baldue"));
        System.out.println("cdtdue: " + rec1.getField("cdtdue"));
```

Come termine di paragone, ecco cme lo stesso record potrebbe essere interpretato utilizzando RFML.

Il codice Java utilizzato per interpretare il contenuto del record dati del cliente tramite RFML potrebbe presentarsi nel modo seguente:

```
// Buffer contenente la rappresentazione binaria di un record di informazioni.
    byte[] bytes;

    // ... Leggere i dati record nel buffer ...

    // Analizzare il file RFML in un oggetto RecordFormatDocument.
// Il file sorgente RFML viene chiamato qcustcdt.rfml.
RecordFormatDocument rfml1 = new RecordFormatDocument("qcustcdt");

    // Leggere il buffer di byte nell'oggetto RecordFormatDocument.
    rfml1.setValues("cusrec", bytes);

    // Richiamare i valori campo.
    System.out.println("cusnum: " + rfml1.getValue("cusrec.cusnum"));
    System.out.println("lstnam: " + rfml1.getValue("cusrec.lstnam"));
    System.out.println("init:   " + rfml1.getValue("cusrec.init"));
    System.out.println("street: " + rfml1.getValue("cusrec.street"));
    System.out.println("city:   " + rfml1.getValue("cusrec.city"));
    System.out.println("state:  " + rfml1.getValue("cusrec.state"));
    System.out.println("zipcod: " + rfml1.getValue("cusrec.zipcod"));
    System.out.println("cdtlmt: " + rfml1.getValue("cusrec.cdtlmt"));
    System.out.println("chgcod: " + rfml1.getValue("cusrec.chgcod"));
    System.out.println("baldue: " + rfml1.getValue("cusrec.baldue"));
System.out.println("cdtdue: " + rfml1.getValue("cusrec.cdtdue"));
```

Classe RecordFormatDocument

La classe RecordFormatDocument consente ai programmi Java di effettuare la conversione tra rappresentazioni di dati RFML ed oggetti Record e RecordFormat per l'utilizzo con altri componenti di IBM Toolbox per Java.

La classe RecordFormatDocument rappresenta un file sorgente RFML e fornisce metodi che consentono al proprio programma Java di eseguire le seguenti azioni:

- Comporre i file sorgente RFML da oggetti Record, oggetti RecordFormat e schiere di byte.
- Creare oggetti Record, oggetti RecordFormat e schiere byte che rappresentano le informazioni contenute nell'oggetto RecordFormatDocument
- Richiamare e impostare i valori dei differenti oggetti e i tipi dati
- Creare XML (RFML) che rappresenta i dati contenuti nell'oggetto RecordFormatDocument
- Serializzare il file sorgente RFML rappresentato dall'oggetto RecordFormatDocument

Per maggiori informazioni sui metodi disponibili, consultare i il riassunto del metodo javadoc per la classe RecordFormatDocument.

Utilizzo della classe RecordFormatDocument con altre classi IBM Toolbox per Java

Utilizzare la classe RecordFormatDocument con le seguenti classi IBM Toolbox per Java:

- Classi orientate a record, comprese le classi file di accesso livello a record (AS400File, SequentialFile eKeyedFile) che leggono, manipolano e scrivono gli oggetti Record. Questa categoria include anche la classe LineDataRecordWriter.
- Classi orientate a byte, comprese alcune classi DataQueue, UserSpace e IFSFile che leggono e scrivono una schiera a byte di dati alla volta.

Non utilizzare la classe RecordFormatDocument con le seguenti classi IBM Toolbox per Java, che leggono e scrivono dati in formati che RecordFormatDocument non gestisce:

- Le classi DataArea poiché i metodi di lettura e scrittura si riferiscono solo a tipi dati String, boolean e BigDecimal.
- IFSTextFileInputStream e IFSTextFileOutputStream perché questi metodi di scrittura e lettura si riferiscono solo a String.
- Le classi JDBC poiché RFML si focalizza solo su dati descritti dalla DDS (data description specification) iSeries.

Documenti formato record e sintassi RFML

I documenti RFML, denominati file sorgente RFML, contengono tag che definiscono la specifica per un formato dati particolare.

Poiché RFML si basa su PCML, gli utenti PCML conoscono già la sintassi. Poiché RFML è un'estensione di XML, è facile leggere e creare i file sorgente RFML. Ad esempio, è possibile creare un file sorgente RFML utilizzando un semplice editor di testo. Inoltre, i file sorgente RFML rivelano la struttura dei dati in modo più comprensibile rispetto ad un linguaggio di programmazione come Java.

L'esempio RFML Utilizzare RFML rispetto ad utilizzare le classi Record di IBM Toolbox per Java include un file sorgente RFML di esempio.

DTD RFML

La DTD (document type definition) RFML definisce gli elementi RFML validi e la sintassi.Per assicurarsi che un programma di analisi XML possa convalidare il proprio file sorgente RFML in fase di esecuzione, dichiarare la DTD RFML nel file sorgente:

```
<!DOCTYPE rfml SYSTEM "rfml.dtd">
```

La DTD RFML risiede nel file jt400.jar (com/ibm/as400/data/rfml.dtd).

Sintassi RFML

La DTD RFML definisce le tag, ognuna delle quali ha le proprie tag attributo. Si utilizzano le tag RFML per dichiarare e definire gli elementi seguenti nei propri file sorgente RFML:

- La tag rfml apre e chiude il file sorgente RFML che descrive il formato dati.
- La tag struct definisce una struttura denominata che è possibile riutilizzare all'interno del file sorgente RFML.La struttura contiene una tag data per ogni campo nella struttura.
- La tag recordformat definisce un formato record, contenente gli elementi dei dati o i riferimenti agli elementi della struttura.
- La tag data definisce un campo all'interno di un formato record o di una struttura.

Nel seguente esempio, la sintassi RFML descrive un formato record e una struttura:

```
<rfml>

    <recordformat>
        <data> </data>
    </recordformat>

    <struct>
        <data> </data>
    </struct>

</rfml>
```

Definizione tipo documento RFML:

Questa è la DTD RFML. Tener presente che la versione è 4.0. La DTD RFML risiede nel file jt400.jar (com/ibm/as400/data/rfml.dtd).

```
<!--
Record Format Markup Language (RFML) Document Type Definition.

RFML is an XML language.  Typical usage:
<?xml version="1.0"?>
   <!DOCTYPE rfml SYSTEM "rfml.dtd">    <rfml version="4.0">
   ...
   </rfml>

(C) Copyright IBM Corporation, 2001,2002
Tutti i diritti riservati.
Licensed Materials Property of IBM
US Government Users Restricted Rights
Use, duplication or disclosure restricted by
GSA ADP Schedule Contract with IBM Corp.
-->

<!-- Entità di convenienza -->
<!ENTITY % string             "CDATA">    <!-- una stringa di lunghezza 0 o maggiore -->
<!ENTITY % nonNegativeInteger "CDATA">    <!-- un numero intero non negativo -->
<!ENTITY % binary2            "CDATA">    <!-- un numero intero compreso nell'intervallo 0-65535 -->
<!ENTITY % boolean            "(true|false)">
<!ENTITY % datatype  "(char | int | packed | zoned | float | byte | struct)">
<!ENTITY % biditype  "(ST4 | ST5 | ST6 | ST7 | ST8 | ST9 | ST10 | ST11 | DEFAULT)">


<!-- L'elemento root del documento -->
<!ELEMENT rfml (struct | recordformat)+>
<!ATTLIST rfml
          version       %string;    #FIXED "4.0"
                    ccsid         %binary2;             #IMPLIED
>
<!-- Nota: il ccsid è il valore predefinito che verrà utilizzato per -->
     <!-- qualsiasi elemento <data type="char"> contenuto senza un ccsid specificato. -->


<!-- Nota: RFML non supporta dichiarazioni struct nidificate.  -->
<!-- Tutti gli elementi struct sono secondari diretti del nodo root. -->
<!ELEMENT struct (data)+>
<!ATTLIST struct
                    name          ID                    #REQUIRED
>


<!-- <!ELEMENT recordformat (data | struct)*> -->
<!ELEMENT recordformat (data)*>
<!ATTLIST recordformat
                    name          ID                    #REQUIRED
                    description   %string;              #IMPLIED
>
<!-- Nota: sul server, il campo "descrizione testo" del Record è limitato a 50 byte. -->

<!ELEMENT data EMPTY>
<!ATTLIST data
                    name          %string;              #REQUIRED

                    count         %nonNegativeInteger;  #IMPLIED

                    type          %datatype;            #REQUIRED
                    length        %nonNegativeInteger;  #IMPLIED
                    precision     %nonNegativeInteger;  #IMPLIED
                    ccsid         %binary2;             #IMPLIED
                    init          CDATA                 #IMPLIED
                    struct        IDREF                 #IMPLIED
```

```
                    bidistringtype %biditype;           #IMPLIED
>
<!-- Nota: l'attributo 'name' deve essere univoco entro un recordformat specificato. -->
<!-- Nota: sul server, la lunghezza dei nomi del campo Record è limitata a 10 byte. -->
<!-- Nota: l'attributo 'length' è necessario, tranne quando type="struct". -->
<!-- Nota: se type="struct", l'attributo 'struct' è necessario. -->
<!-- Nota: gli attributi 'ccsid' e 'bidistringtype' sono validi solo quando type="char". -->
<!-- Nota: l'attributo 'precision' è valido solo per tipi "int", "packed" e "zoned". -->


<!-- Le entità carattere predefinite standard -->
<!ENTITY quot  "&#34;">     <!-- virgolette -->
<!ENTITY amp   "&#38;#38;"> <!-- ampersand -->
<!ENTITY apos  "&#39;">     <!-- apostrofo -->
<!ENTITY lt    "&#38;#60;"> <!-- minore di -->
<!ENTITY gt    "&#62;">     <!-- maggiore di -->
<!ENTITY nbsp  " ">    <!-- spazio senza interruzione -->
<!ENTITY shy   "&#173;">    <!-- trattino sfumato (trattino di separazione) -->
<!ENTITY mdash "&#38;#x2014;">
<!ENTITY ldquo "&#38;#x201C;">
<!ENTITY rdquo "&#38;#x201D;">
```

La tag Data RFML:

La tag data può avere i seguenti attributi. Gli attributi racchiusi tra parentesi, [], indicano che l'attributo è facoltativo. Se si specifica un attributo facoltativo, non includere le parentesi nel proprio sorgente. Alcuni valori di attributo vengono visualizzati come un elenco di opzioni racchiuse tra parentesi graffe, {}, con le scelte possibili separate da barre verticali, |. Quando si specifica uno di questi attributi, non includere le parentesi graffe nel sorgente e specificare solo una delle scelte visualizzate.

```
<data type="{ char | int | packed | zoned | float | byte | struct }" ]
    [ bidistringtype="{ ST4 | ST5 | ST6 | ST7 | ST8 | ST9 | ST10 | ST11 | DEFAULT }"]
    [ ccsid="{ number | data-name }" ]
    [ count="{ number | data-name }" ]
    [ init="string" ]
    [ length="{ number | data-name }" ]
    [ name="name" ]
    [ precision="number" ]
    [ struct="struct-name" ]>
</data>
```

La seguente tabella elenca gli attributi della tag Data. Ogni voce include il nome dell'attributo, i possibili valori validi e una descrizione dell'attributo

Attributo	Valore	Descrizione
type=	*char* Un valore carattere. Un valore dati *char* viene restituito come un *java.lang.String*.Per ulteriori informazioni, consultare i *valori char* per length. *int* Un valore numero intero. Un valore dati *int* viene restituito come un *java.lang.Long*. Per ulteriori informazioni, consultare i valori *int* per length e precision. *packed* Un valore decimale compresso. Un valore dati *packed* viene restituito come un *java.math.BigDecimal*.Per ulteriori informazioni, consultare i *valori packed* per length e precision. *zoned* Un valore A zonatura decimale. Un valore dati *zoned* viene restituito come un *java.math.BigDecimal*.Per ulteriori informazioni, consultare i *valori zoned* per length e precision. *float* Un valore a virgola mobile. L'attributo **length** specifica il numero di byte: `4` o `8`. Un numero intero a 4-byte è restituito come *java.lang.Float*. Un numero intero a 8 byte viene restituito come un *java.lang.Double*.Per ulteriori informazioni, consultare i *valori float* per length. *byte* Un valore byte. Non viene eseguita alcuna conversione sui dati. Un valore dati *byte* viene restituito come una schiera di valori *byte* (*byte[]*).Per ulteriori informazioni, consultare i *valori byte* per length. *struct* Il nome dell'elemento **<struct>**. Un elemento *struct* consente di definire una struttura una volta e di riutilizzarla più volte all'interno del documento.Quando si utilizza **type=**`"struct"`, è come se la struttura specificata venisse visualizzata in questa ubicazione nel documento.Un elemento *struct* non prevede un valore length e non ha un valore per precision.	Indica il tipo di dati che si sta utilizzando (char, integer, packed, zoned, floating point, byte o struct). I valori degli attributi length e precision sono differenti per diversi tipi di dati. Per ulteriori informazioni, consultare i valori per length e precision.

Attributo	Valore	Descrizione
bidistringtype=	*DEFAULT* dove DEFAULT è il tipo stringa predefinito per dati non bidirezionali (LTR). *ST4* dove *ST4* è Tipo stringa 4. *ST5* dove *ST5* è Tipo stringa 5. *ST6* dove *ST6* è Tipo stringa 6. *ST7* dove *ST7* è Tipo stringa 7. *ST8* dove *ST8* è Tipo stringa 8. *ST9* dove *ST9* è Tipo stringa 9. *ST10* dove *ST10* è Tipo stringa 10. *ST11* dove *ST11* è Tipo stringa 11.	Specifica il tipo stringa bidirezionale per gli elementi **<data>** con **type=**″char″. Se questo attributo viene omesso, il tipo di stringa per questo elemento è contenuto nel CCSID (se esplicitamente specificato o se il CCSID predefinito dell'ambiente host). I tipi di stringa sono definiti nella Javadoc per la classe BidiStringType.
ccsid=	*number* dove *number* definisce un CCSID fisso, invariabile. *data-name* dove *data-name* definisce il nome che conterrà, in fase di esecuzione, il CCSID dei dati carattere. E' possibile che *data-name* specificato sia un nome completo o un nome relativo all'elemento corrente. In entrambi i casi, il nome deve riferirsi ad un elemento **<data>** che viene definito con **type=**″int″. Consultare Definizione dei nomi relativi per ulteriori informazioni sulle modalità di definizione dei nomi relativi.	Specifica il CCSID (coded character set identifier) dell'host per dati carattere relativi all'elemento **<data>**. L'attributo **ccsid** può essere specificato solo per elementi **<data>** con **type=**″char″. Se questo attributo viene omesso, si presume che i dati carattere per questo elemento siano nel CCSID predefinito dell'ambiente host.
count=	*number* dove *number* definisce un numero fisso, invariabile di elementi in una schiera dimensionata. *data-name* dove *data-name* definisce il nome di un elemento **<data>** nel documento RFML che conterrà, in fase di esecuzione, il numero di elementi nella schiera. E' possibile che *data-name* specificato sia un nome completo o un nome relativo all'elemento corrente. In entrambi i casi, il nome deve riferirsi ad un elemento **<data>** che viene definito con **type=**″int″. Consultare Definizione dei nomi relativi per ulteriori informazioni sulle modalità di definizione dei nomi relativi.	Specifica se l'elemento è una schiera e identifica il numero di voci nella schiera. Se l'attributo `count` viene omesso, l'elemento non è definito come una schiera, anche se è possibile che sia contenuto all'interno di un altro elemento definito come una schiera.

Attributo	Valore	Descrizione
init=	*string*	Specifica un valore iniziale per l'elemento **<data>**. Si utilizza il valore iniziale specificato per inizializzare i valori scalari. Se l'elemento è definito come una schiera o è contenuto all'interno di una struttura definita come una schiera, il valore iniziale specificato viene utilizzato come un valore iniziale per tutte le voci nella schiera.
length=	*number* dove *number* definisce una lunghezza fissa, invariabile. *data-name* dove *data-name* definisce il nome di un elemento **<data>** entro il documento RFML che conterrà, in fase di esecuzione, la lunghezza. Un *data-name* può essere specificato solo per elementi **<data>** con **type=**"char" o **type=**"byte". E' possibile che *data-name* specificato sia un nome completo o un nome relativo all'elemento corrente. In entrambi i casi, il nome deve riferirsi ad un elemento **<data>** che viene definito con **type=**"int". Consultare Definizione dei nomi relativi per ulteriori informazioni sulle modalità di definizione dei nomi relativi.	Specifica la lunghezza dell'elemento data. La variazione nell'utilizzo di questo attributo dipende dal tipo di dati. Per ulteriori informazioni, consultare i valori per length e precision.
name=	*name*	Specifica il nome dell'elemento **<data>**.
precision=	*number*	Specifica il numero di byte di precisione per alcuni tipi di dati numerici. Per ulteriori informazioni, consultare i valori per length e precision.
struct=	*name*	Specifica il nome di un elemento **<struct>** per l'elemento **<data>**. Un attributo **struct** può essere specificato solo per elementi **<data>** con **type=**"struct".

La tag rfml RFML:

La tag rfml può avere i seguenti attributi. Gli attributi racchiusi tra parentesi, [], indicano che l'attributo è facoltativo. Se si specifica un attributo facoltativo, non includere le parentesi nel proprio sorgente.

```
<rfml version="version-string"
    [ ccsid="number" ]>
</rfml>
```

La tabella seguente indica gli attributi della tag rfml. Ogni voce include il nome dell'attributo, i possibili valori validi e una descrizione dell'attributo

Attributo	Valore	Descrizione
version=	*version-string* Una versione fissa della DTD RFML. Per V5R3, `4.0` è il solo valore valido.	Specifica la versione della DTD RFML, che è possibile utilizzare per verificare il valore corretto.
ccsid=	*number* Un CCSID (coded character set identifier) fisso, invariabile.	Specifica il CCSID dell'host, che si applica a tutti gli elementi racchiusi in <**data type**="char"> che non specifica un CCSID. Per ulteriori informazioni, consultare la tag **<data>** RFML. Se si omette questo attributo, si utilizza il CCSID predefinito dell'ambiente host.

La tag recordformat RFML:

La tag recordformat può avere i seguenti attributi. Gli attributi racchiusi tra parentesi, [], indicano che l'attributo è facoltativo. Se si specifica un attributo facoltativo, non includere le parentesi nel proprio sorgente.

```
<recordformat name="name"
    [ description="description" ]>
</recordformat>
```

La seguente tabella elenca gli attributi della tag recordformat. Ogni voce include il nome dell'attributo, i possibili valori validi e una descrizione dell'attributo

Attributo	Valore	Descrizione
name=	*name*	Specifica il nome del recordformat.
description=	*description*	Specifica la descrizione del recordformat.

La tag struct RFML:

La tag struct può avere i seguenti attributi. Gli attributi racchiusi tra parentesi, [], indicano che l'attributo è facoltativo. Se si specifica un attributo facoltativo, non includere le parentesi nel proprio sorgente.

```
<struct name="name">
</struct>
```

La seguente tabella elenca gli attributi della tag struct. Ogni voce include il nome dell'attributo, i possibili valori validi e una descrizione dell'attributo

Attributo	Valore	Descrizione
name=	*name*	Specifica il nome dell'elemento **<struct>**.

Programma di analisi XML e processore XSLT

Alcuni pacchetti o funzioni di IBM Toolbox per Java richiedono che, al tempo di esecuzione, si disponga di un programma di analisi XML (Extensible Markup Language) o di un processore XSLT (Extensible Stylesheet Language Transformations) nella variabile di ambiente CLASSPATH. Fare riferimento alle seguenti informazioni per quale programma di analisi e processore si desidera utilizzare.

Per ulteriori informazioni su quali pacchetti e funzioni di IBM Toolbox per Java richiedono un programma di analisi XML o un processore XSLT, consultare la seguente pagina:

"File Jar" a pagina 14

Programma di analisi XML

Se il pacchetto o la funzione richiede un programma di analisi XML, è necessario includerne uno in CLASSPATH al tempo di esecuzione. Il programma di analisi XML deve soddisfare i seguenti requisiti:

- Essere compatibile con JAXP
- Estendere la classe org.apache.xerces.parsers.SAXParser
- Supporto convalida completa dello schema

Nota: il programma di analisi deve supportare la convalida completa dello schema solo se l'utente intende utilizzare XPCML. Se si sta utilizzando solo PCML, la convalida completa dello schema non è necessaria.

IL J2SDK (Java 2 Software Developer Kit), versione 1.4, include un programma di analisi XML idoneo. Se si utilizza una versione precedente di J2SDK, utilizzare uno qualsiasi dei seguenti metodi per accedere ad un programma di analisi XML idoneo:

- Utilizzare x4j400.jar (una versione IBM del programma di analisi Xerces XML da Apache)
- Scaricare il programma di analisi Xerces XML dal sito Web Apache
- Utilizzare qualsiasi programma di analisi XML compatibile nell'indirizzario /QIBM/ProdData/OS400/xml/lib sul server iSeries

Nota: considerare l'utilizzo delle più recenti versioni dei programmi di analisi che risiedono in /QIBM/ProdData/OS400/xml/lib. Ad esempio, xmlapis11.jar e xerces411.jar sono entrambi programmi di analisi a convalida completa.

E' possibile utilizzare questi programmi di analisi sul server o copiarli su una stazione di lavoro.

Nota: qualsiasi programma di analisi XML che soddisfi i requisiti per l'esecuzione di XPCML può eseguire PCML e RFML. Tenere a mente che XPCML non supporta la serializzazione.

Processore XSLT

Se il pacchetto o la funzione richiede un processore XSLT, è necessario includerne uno in CLASSPATH al tempo di esecuzione. Il processore XSLT deve soddisfare i seguenti requisiti:

- Essere compatibile con JAXP
- Contenere la classe javax.xml.transform.Transformer

Il J2SDK (Java 2 Software Developer Kit), versione 1.4, include un processore XSLT idoneo. Se si utilizza una precedente versione di J2SDK, utilizzare uno qualsiasi dei seguenti metodi per accedere ad un processore XSLT idoneo:

- Utilizzare xslparser.jar (una versione IBM del processore Xalan XSLT da Apache)
- Scaricare il processore Xalan XSLT dal sito Web Apache
- Utilizzare qualsiasi processore XSLT compatibile nell'indirizzario /QIBM/ProdData/OS400/xml/lib sul server iSeries

E' possibile utilizzare questi processori sul server o copiarli su una stazione di lavoro.

XPCML (Extensible Program Call Markup Language)

XPCML (Extensible Program Call Markup Language) potenzia la funzionalità e la capacità di utilizzo di PCML (Program Call Markup Language) fornendo il supporto per schemi XML. XPCML non supporta la serializzazione, così a differenza di PCML, non è possibile serializzare un documento XPCML. Tuttavia, XPCML offre diversi validi potenziamenti rispetto a PCML:

- Specificare e passare valori per parametri di programma
- Richiamare i risultati di una chiamata di programma nel server iSeries in XPCML
- Trasformare un documento PCML esistente nell'equivalente documento XPCML
- Estendere e personalizzare lo schema XPCML per definire nuovi elementi ed attributi semplici e complessi

Per ulteriori informazioni su XPCML, consultare le seguenti pagine:

Vantaggi di XPCML rispetto a PCML
Acquisire informazioni sui validi potenziamenti di XPCML rispetto a PCML.

Requisiti
Esaminare i requisiti software per l'utilizzo di XPCML.

Schema e sintassi XPCML
Verificare come lo schema XPCML definisce la sintassi XPCML ed i tipi di dati server disponibili. Acquisire informazioni su come estendere e personalizzare lo schema per soddisfare le esigenze di programmazione.

Utilizzo di XPCML
Acquisire informazioni su come avvantaggiarsi dei potenziamenti di XPCML, inclusa la trasmissione di differenti tipi di parametri al programma server ed il richiamo dei dati di parametro restituiti. Apprendere come condensare il codice XPCML, che rende il codice più facile da utilizzare e leggere e come utilizzare XPCML con le applicazioni correnti abilitate a PCML.

XPCML è l'unico modo di utilizzare XML con il server. Per ulteriori informazioni sull'utilizzo di XML, consultare le seguenti pagine:

Componenti XML (Extensible Markup Language)

XML Toolkit

W3C Architecture domain: XML Schema 

Vantaggi di XPCML rispetto a PCML

XPCML (Extensible Program Call Markup Language) offre diversi validi potenziamenti rispetto a PCML:
- Specifica ed inoltro di valori per parametri di programma
- Richiamare i risultati di una chiamata di programma nel server iSeries in XPCML
- Trasformare un documento PCML esistente nell'equivalente documento XPCML
- Estendere e personalizzare lo schema XPCML per definire nuovi elementi ed attributi semplici e complessi

Specifica ed inoltro di valori per parametri di programma

XPCML utilizza uno schema XML per definire i tipi di parametri di programma; PCML utilizza un DTD (data type definition). In fase di analisi, il programma di analisi XML convalida i valori dei dati immessi come parametri rispetto ai parametri appropriati come sono stati definiti nello schema. Esistono tipi di dati per parametri di molti tipi: string, integer, long e così via. Questa capacità di specificare e passare valori per i parametri di programma è un miglioramento significativo rispetto a PCML. In PCML, è possibile verificare i valori per i parametri solo dopo l'analisi del documento PCML. Inoltre, la verifica dei valori di parametro in PCML spesso richiede la codifica dell'applicazione per eseguire la convalida.

Richiamo dei risultati di una chiamata al programma in XPCML

XPCML inoltre fornisce la capacità di richiamare i risultati di una chiamata al programma come XPCML. In PCML, si ottengono i risultati di una chiamata al programma chiamando uno dei metodi getValue della classe ProgramCallDocument dopo aver effettuato la chiamata al programma. In XPCML, è

possibile utilizzare i metodi getValue, ma è anche possibile fare sì che il proprio XPCML chiami un metodo generateXPCML, che restituisce i risultati di una chiamata al programma come XPCML.

Trasformazione di documenti PCML esistenti in XPCML

Un nuovo metodo della classe ProgramCallDocument, transformPCMLToXPCML, consente di trasformare i documenti PCML esistenti in documenti XPCML equivalenti. Questo consente di avvantaggiarsi della nuova funzione XPCML senza scrivere un sorgente XPCML per i documenti di chiamata al programma iSeries esistenti.

Estensione e personalizzazione dello schema XPCML

XPCML è estensibile il che significa che è possibile definire nuovi tipi di parametri che estendono quelli specificati dallo schema XPCML. Condensando XPCML si estende lo schema XPCML in modo da creare nuove DTD (data type definition) che semplificano e migliorano la capacità di leggere e utilizzare i documenti XPCML.

Requisiti per l'utilizzo di XPCML

L'XPCML (Extensible Program Call Markup Language) ha gli stessi requisiti JVM stazione di lavoro del resto di IBM Toolbox per Java. Per ulteriori informazioni, consultare la seguente pagina:

"Requisiti stazione di lavoro per l'esecuzione di applicazioni IBM Toolbox per Java" a pagina 11

Inoltre, l'utilizzo di XPCML include requisiti per il file di schema XML, il programma di analisi XML ed il processore XSLT (Extensible Stylesheet Language Transformation):

File di schema XML

I documenti XPCML devono riconoscere l'ubicazione del file che contiene lo schema. Lo schema predefinito per IBM Toolbox per Java è xpcml.xsd, che risiede nel file jt400.jar.

Per utilizzare lo schema predefinito, estrarre xpcml.xsd da jt400.jar, quindi ubicare il file in un'ubicazione idonea. La seguente procedura mostra una modalità di estrazione del file .xsd in una stazione di lavoro.

Estrazione del file di schema xpcml.xsd

- Avviare una sessione della riga comandi nell'indirizzario che contiene jt400.jar
- Utilizzare il seguente comando per estrarre il file .xsd:

```
jar xvf  jt400.jar   com/ibm/as400/data/xpcml.xsd
```

Nota: se non si esegue il comando precedente da un indirizzario che contiene jt400.jar, è possibile specificare un percorso completo per jt400.jar.

è possibile ubicare collocare il file di schema predefinito (o qualsiasi file di schema) in qualunque indirizzario. Il solo requisito è quello di specificare l'ubicazione del file di schema utilizzando l'attributo xsi:noNamespaceSchemaLocation nella tag <xpcml>.

E' possibile specificare l'ubicazione dello schema come percorso file o come URL.

Nota: sebbene i seguenti esempi utilizzino xpcml.xsd come file di schema, è possibile specificare qualsiasi schema che estenda xpcml.xsd.

- Per specificare lo stesso indirizzario del file XPCML, utilizzare xsi:noNamespaceSchemaLocation='xpcml.xsd'
- Per specificare un percorso completo: xsi:noNamespaceSchemaLocation='c:\myDir\xpcml.xsd'
- Per specificare un URL: xsi:noNamespaceSchemaLocation='http://myServer/xpcml.xsd'

Per visualizzare una versione HTML del file xpcml.xsd, consultare la seguente pagina:

"File Schema xpcml.xsd" a pagina 423

Programma di analisi XML e processore XSLT

Al tempo di esecuzione, è necessario includere un programma di analisi XML ed un processore XSLT nella variabile d'ambiente CLASSPATH. Per ulteriori informazioni, consultare la seguente pagina:

"Programma di analisi XML e processore XSLT" a pagina 417

Schema e sintassi XPCML

I documenti XPCML, denominati file sorgente XPCML, contengono tag e dati che definiscono completamente chiamate a programmi sul server iSeries.

Dal momento che XPCML utilizza schemi XML invece di una DTD (document type definition), è possibile utilizzare XPCML in modi in cui non è possibile utilizzare PCML:

- Trasmettere valori per i parametri di immissione al programma come elementi XML
- Ricevere valori per i parametri di emissione dal programma come elementi XML
- Fare sì che il programma di analisi XML convalidi automaticamente i valori trasmessi al programma
- Estendere lo schema per definire nuovi elementi semplici e complessi

Per ulteriori informazioni sullo schema e la sintassi XPCML, consultare le seguenti pagine:

Confronto tra sorgente XPCML e sorgente PCML
Esaminare esempi che mettono a confronto sorgente XPCML e sorgente PCML. Gli esempi illustrano come XPCML fornisce maggiore funzionalità e facilita la lettura e la scrittura dei dati sorgente.

Schema XPCML
Esaminare il file di schema XPCML e acquisire maggiori informazioni sull'utilizzo e l'estensione dello schema XPCML.

Sintassi XPCML
Esaminare un elenco di elementi di sintassi XPCML che lo schema utilizza per definire gli elementi XPCML.

Attributi tag XPCML
Descrizione degli attributi differenti per ogni elemento che lo schema XPCML definisce.

Confronto tra sorgente XPCML e sorgente PCML:

XPCML differisce da PCML in diversi aspetti, ma una delle principali differenze è che XPCML consente di specificare i valori di parametri di immissione all'interno del file sorgente XPCML.

PCML consente di utilizzare l'attributo init della tag <data> per specificare il valore iniziale per un elemento dati nel sorgente PCML. Tuttavia, l'utilizzo di PCML per specificare valori ha i seguenti limiti:

- Non è possibile utilizzare l'attributo init per impostare valori di schiera
- La convalida del valore init avviene solo dopo l'analisi del documento PCML

Per specificare i valori di schiera in PCML, è necessario prima leggere ed analizzare il documento PCML, quindi eseguire una serie di chiamate a ProgramCallDocument.setValue().

L'utilizzo di XPCML rende più facile specificare valori di singoli elementi e schiere:

- Specificare valori per elementi sia scalari che di schiera nel file sorgente XPCML
- Convalidare i valori di schiera specificati in fase di analisi

I seguenti confronti semplici indicano modi in cui XPCML differisce da PCML. Ogni esempio definisce una chiamata di programma per un programma server iSeries.

Esempio: chiamata ad un programma server iSeries

I seguenti esempi chiamano un programma server iSeries denominato prog1.

Codice sorgente XPCML

```
<?xml version="1.0" encoding="UTF-8"?>
<xpcml xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:noNamespaceSchemaLocation='xpcml.xsd' version="4.0">
    <program name="prog1" path="/QSYS.LIB/W95LIB.LIB/PROG1.PGM">
      <parameterList>
         <stringParm name="parm1"  passDirection="in" length="10">Parm1</stringParm>
         <intParm    name="parm2"  passDirection="in">5</intParm>
         <shortParm  name="parm3"  passDirection="in">3</shortParm>
      </parameterList>
    </program>
</xpcml>
```

Codice sorgente PCML

```
<pcml version="4.0">
  <program name="prog1" path="QSYS.LIB/MYLIB.LIB/PROG1.PGM">
     <data name="parm1"  type="char" usage="input" length="10" init="Parm1"/>
     <data name="parm2"  type="int"  usage="input" length="4"  init="5"/>
     <data name="parm3"  type="int"  usage="input" length="2"  precision="16" init="3"/>
  </program>
</pcml>
```

Esempio: chiamata ad un programma server iSeries utilizzando una schiera di parametri string

I seguenti esempi chiamano un programma server iSeries denominato prog2 e definiscono parm1 come una schiera di parametri string. Si noti la funzionalità di XPCML:

- Inizializza il valore di ogni elemento nella schiera
- Specifica i valori di immissione come contenuto dell'elemento che un programma di analisi XML a convalida completa può verificare

E' possibile avvantaggiarsi di questa funzionalità XPCML senza scrivere alcun codice Java.

PCML non può avere prestazioni corrispondenti a quelle di XPCML. PCML non può inizializzare il valore di ogni elemento nella schiera. PCML non può convalidare i valori init in fase di analisi. Per avere una funzionalità corrispondente a quella di XPCML, si dovrebbe leggere ed analizzare il documento PCML, quindi codificare l'applicazione Java per impostare il valore relativo ad ogni elemento di schiera. Si dovrebbe anche scrivere un codice per convalidare i parametri.

Codice sorgente XPCML

```
<?xml version="1.0" encoding="UTF-8"?>
<xpcml xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:noNamespaceSchemaLocation="xpcml.xsd" version="4.0">

    <program name="prog2" path="/QSYS.LIB/W95LIB.LIB/PROG2.PGM">
      <parameterList>
         <arrayOfStringParm name="parm1" passDirection="in"
            length="10" count="3">
             <i>Parm1-First value</i>
             <i>Parm1-Second value</i>
             <i>Parm1-Third value</i>
         </arrayOfStringParm>
         <longParm name="parm2"  passDirection="in">5</longParm>
```

```
            <zonedDecimalParm name="parm3"  passDirection="in"
               totalDigits="5" fractionDigits="2">32.56</zonedDecimalParm>
          </parameterList>
        </program>
    </xpcml>
```

Codice sorgente PCML

```
    <pcml version="4.0">
      <program name="prog2" path="QSYS.LIB/MYLIB.LIB/PROG2.PGM">
         <data name="parm1"  type="char"  usage="input" length="20" count="3"/>
         <data name="parm2"  type="int"   usage="input" length="8"  init="5"/>
         <data name="parm3"  type="zoned" usage="input" length="5"  precision="2" init="32.56"/>
      </program>
    </pcml>
```

File Schema xpcml.xsd:

Per ulteriori informazioni sull'utilizzo del file xpcml.xsd, consultare Requisiti per l'utilizzo di XPCML.

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

Per facilitare la visualizzazione e la stampa, alcune righe di questa versione HTML di xpcml.xsd vanno a capo su una seconda riga. Le stesse righe nel file xsd sorgente vengono visualizzate su un'unica riga singola.

```
<?xml version="1.0" encoding="UTF-8"?>

<!--////////////////////////////////////////////////////////////////////////
//
// JTOpen (IBM Toolbox per Java - versione OSS)
//
// Nome file: xpcml.xsd
//
// Il codice sorgente qui contenuto è su licenza di IBM Public License
// Versione 1.0, approvata da Open Source Initiative.
// Copyright (C) 1997-2003 International Business Machines Corporation ed
// altri. Tutti i diritti riservati.
//
////////////////////////////////////////////////////////////////////////-->

<xs:schema xmlns:xs='http://www.w3.org/2001/XMLSchema'>

<xs:annotation>
 <xs:documentation>
  Schema for xpcml (eXtended Program Call Markup Language).
 </xs:documentation>
</xs:annotation>

<xs:element name="xpcml">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="structOrProgram" minOccurs="1" maxOccurs="unbounded" />
    </xs:sequence>
    <xs:attribute name="version" use="required">
     <xs:simpleType>
       <xs:restriction base="xs:string">
         <xs:enumeration value="4.0"/>
       </xs:restriction>
     </xs:simpleType>
   </xs:attribute>
  </xs:complexType>

  <!-- Definire il collegamento key/keyref tra il nome di struct   -->
  <!-- e l'attributo struct di un campo di parametro.        -->
  <xs:key name="StructKey">
```

```
    <xs:selector xpath="struct"/>
    <xs:field xpath="@name"/>
  </xs:key>
  <xs:keyref name="spRef" refer="StructKey">
     <xs:selector xpath="structParm" />
     <xs:field xpath="@struct" />
  </xs:keyref>
</xs:element>

<!-- Tag program e attributi -->
<xs:element name="program" substitutionGroup="structOrProgram">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="parameterList" minOccurs="1" maxOccurs="1"/>
         <!-- Utilizzata come tag per il riavvolgimento ciclico dell'elenco di parametri per il programma. -->
    </xs:sequence>
    <!-- Nome del programma da chiamare. -->
    <xs:attribute name="name" type="string50" use="required" />
    <!-- Percorso per l'oggetto programma. L'impostazione predefinita è presupporlo nella libreria QSYS. -->
    <xs:attribute name="path" type="xs:string"/>
    <!-- Specifica l'ordine in cui parametri dovrebbero essere analizzati. -->
    <!-- Il valore è un elenco separato da spazi di nomi di parametro. -->
    <xs:attribute name="parseOrder" type="xs:string"/>
    <!-- Il nome del punto di immissione in un programma di servizio. -->
    <xs:attribute name="entryPoint" type="xs:string"/>
    <!-- Il tipo di valore, se esiste, restituito da una chiamata al programma di servizio. -->
    <xs:attribute name="returnValue" type="returnValueType"/>
    <!-- Quando la chiamata ad un programma Java e ad un programma iSeries è sullo stesso server -->
    <!-- ed è thread-safe, impostare il valore su true per chiamare il programma iSeries nello stesso lavoro -->
    <!-- e nello stesso sottoprocesso del programma Java. -->
    <xs:attribute name="threadSafe" type="xs:boolean"  />
    <!-- Il CCSID del nome punto di immissione in un programma di servizio. -->
    <xs:attribute name="epccsid" type="ccsidType"/>
  </xs:complexType>
</xs:element>

<!-- Un elenco di parametri è composto di uno o più parametri. -->
<xs:element name="parameterList">
  <xs:complexType>
      <xs:group ref="programParameter" minOccurs="1" maxOccurs="unbounded"/>
   </xs:complexType>
</xs:element>

<!-- Tutti i differenti tipi di parametri di programma riconosciuti. -->
<xs:group name="programParameter">
      <xs:choice>
 <xs:element ref="stringParmGroup"/>
 <xs:element ref="stringParmArrayGroup"/>
 <xs:element ref="intParmGroup"/>
 <xs:element ref="intParmArrayGroup"/>
        <xs:element ref="unsignedIntParmGroup"/>
        <xs:element ref="unsignedIntParmArrayGroup"/>
 <xs:element ref="shortParmGroup"/>
 <xs:element ref="shortParmArrayGroup"/>
 <xs:element ref="unsignedShortParmGroup"/>
 <xs:element ref="unsignedShortParmArrayGroup"/>
 <xs:element ref="longParmGroup"/>
 <xs:element ref="longParmArrayGroup"/>
 <xs:element ref="zonedDecimalParmGroup"/>
 <xs:element ref="zonedDecimalParmArrayGroup"/>
 <xs:element ref="packedDecimalParmGroup"/>
        <xs:element ref="packedDecimalParmArrayGroup"/>
 <xs:element ref="floatParmGroup"/>
 <xs:element ref="floatParmArrayGroup"/>
 <xs:element ref="doubleParmGroup"/>
 <xs:element ref="doubleParmArrayGroup"/>
 <xs:element ref="hexBinaryParmGroup"/>
```

```
 <xs:element ref="hexBinaryParmArrayGroup"/>
        <xs:element ref="structParmGroup"/>
        <xs:element ref="structParmArrayGroup"/>
        <xs:element ref="structArrayGroup"/>
        <xs:element ref="struct"/>
      </xs:choice>

</xs:group>

<!-- Tipo astratto per tutti i tipi di parametri dati. -->
<xs:element name="stringParmGroup" type="stringParmType" abstract="true" />
<xs:element name="stringParmArrayGroup" type="stringParmArrayType" abstract="true" />
<xs:element name="intParmGroup" type="intParmType" abstract="true" />
<xs:element name="intParmArrayGroup" type="intParmArrayType" abstract="true" />
<xs:element name="unsignedIntParmGroup" type="unsignedIntParmType" abstract="true" />
<xs:element name="unsignedIntParmArrayGroup" type="unsignedIntParmArrayType" abstract="true" />
<xs:element name="shortParmGroup" type="shortParmType" abstract="true" />
<xs:element name="shortParmArrayGroup" type="shortParmArrayType" abstract="true" />
<xs:element name="unsignedShortParmGroup" type="unsignedShortParmType" abstract="true" />
<xs:element name="unsignedShortParmArrayGroup" type="unsignedShortParmArrayType" abstract="true" />
<xs:element name="longParmGroup" type="longParmType" abstract="true" />
<xs:element name="longParmArrayGroup" type="longParmArrayType" abstract="true" />
<xs:element name="zonedDecimalParmGroup" type="zonedDecimalParmType" abstract="true" />
<xs:element name="zonedDecimalParmArrayGroup" type="zonedDecimalParmArrayType" abstract="true" />
<xs:element name="packedDecimalParmGroup" type="packedDecimalParmType" abstract="true" />
<xs:element name="packedDecimalParmArrayGroup" type="packedDecimalParmArrayType" abstract="true" />
<xs:element name="floatParmGroup" type="floatParmType" abstract="true" />
<xs:element name="floatParmArrayGroup" type="floatParmArrayType" abstract="true" />
<xs:element name="doubleParmGroup" type="doubleParmType" abstract="true" />
<xs:element name="doubleParmArrayGroup" type="doubleParmArrayType" abstract="true" />
<xs:element name="hexBinaryParmGroup" type="hexBinaryParmType" abstract="true" />
<xs:element name="hexBinaryParmArrayGroup" type="hexBinaryParmArrayType" abstract="true" />
<xs:element name="structParmGroup" type="structParmType" abstract="true" />
<xs:element name="structParmArrayGroup" type="structParmArrayType" abstract="true" />
<xs:element name="structArrayGroup" type="structArrayType" abstract="true"
            substitutionGroup="structOrProgram" />

<!-- Parametro String -->
   <xs:element name="stringParm" type="stringParmType" substitutionGroup="stringParmGroup"
               nillable="true"/>
     <xs:complexType name="stringParmType">
       <xs:simpleContent>
          <xs:extension base="stringFieldType">
            <xs:attributeGroup ref="commonParmAttrs"/>
          </xs:extension>
       </xs:simpleContent>
   </xs:complexType>

<!-- Schiera di parametri string -->
  <xs:element name="arrayOfStringParm" type="stringParmArrayType"
              substitutionGroup="stringParmArrayGroup" nillable="true" />
  <xs:complexType name="stringParmArrayType">
    <xs:sequence>
      <xs:element name="i" type="stringElementType" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attributeGroup ref="commonParmAttrs"/>
    <xs:attributeGroup ref="commonFieldAttrs"/>
    <!-- Il numero di elementi nella schiera. -->
    <!-- 'count' è necessario se si desidera immettere e/o emettere dati di schiera in formato XPCML. -->
    <xs:attribute name="count" type="xs:string" />
    <!-- Il numero di caratteri in ogni stringa. -->
    <xs:attribute name="length" type="xs:string"/>
    <!-- Il CCSID host per ogni stringa. -->
    <xs:attribute name="ccsid" type="xs:string"/>
    <!-- Specifica come adattare lo spazio bianco (sinistra, destra, entrambi, nessuno). -->
    <xs:attribute name="trim" type="trimType" />
    <!-- La dimensione di ogni carattere ('chartype' in PCML). -->
```

```
    <xs:attribute name="bytesPerChar" type="charType" />
   <!-- Il tipo string bidirezionale. -->
    <xs:attribute name="bidiStringType" type="bidiStringTypeType" />
  </xs:complexType>

      <xs:complexType name="stringElementType">
        <xs:simpleContent>
           <xs:extension base="xs:string">
                <!-- L'indice nella schiera. -->
                <xs:attribute name="index" type="xs:nonNegativeInteger"  />
           </xs:extension>
        </xs:simpleContent>
       </xs:complexType>


<!-- Parametro Integer (4 byte sul server) -->
    <xs:element name="intParm" type="intParmType"  nillable="true" substitutionGroup="intParmGroup"   />
      <xs:complexType name="intParmType" >
         <xs:simpleContent>
            <xs:extension base="intFieldType">
              <xs:attributeGroup ref="commonParmAttrs"/>
            </xs:extension>
         </xs:simpleContent>
      </xs:complexType>

<!-- tipo schiera intParmray -->
  <xs:element name="arrayOfIntParm" type="intParmArrayType" substitutionGroup="intParmArrayGroup"
              nillable="true" />
  <xs:complexType name="intParmArrayType">
    <xs:sequence>
      <!-- 'i' è una tag utilizzata per elementi di schiera non struct. -->
      <xs:element name="i" type="intElementType" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <!-- 'count' è necessario se si desidera immettere e/o emettere dati di schiera in formato XPCML. -->
    <xs:attribute name="count" type="xs:string" />
    <xs:attributeGroup ref="commonParmAttrs"/>
    <xs:attributeGroup ref="commonFieldAttrs"/>
  </xs:complexType>

  <xs:complexType name="intElementType">
    <xs:simpleContent>
       <xs:extension base="xs:int">
            <xs:attribute name="index" type="xs:nonNegativeInteger"  />
       </xs:extension>
    </xs:simpleContent>
  </xs:complexType>


<!-- Parametro Integer senza segno (4 byte sul server) -->
    <xs:element name="unsignedIntParm" type="unsignedIntParmType" nillable="true"
                substitutionGroup="unsignedIntParmGroup"   />
      <xs:complexType name="unsignedIntParmType">
         <xs:simpleContent>
            <xs:extension base="unsignedIntFieldType">
              <xs:attributeGroup ref="commonParmAttrs"/>
            </xs:extension>
         </xs:simpleContent>
      </xs:complexType>

<!-- tipo schiera intParm senza segno -->
  <xs:element name="arrayOfUnsignedIntParm" type="unsignedIntParmArrayType"
              substitutionGroup="unsignedIntParmArrayGroup" nillable="true" />
  <xs:complexType name="unsignedIntParmArrayType">
    <xs:sequence>
      <xs:element name="i" type="unsignedIntElementType" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <!-- 'count' è necessario se si desidera immettere e/o emettere dati di schiera in formato XPCML. -->
```

```
    <xs:attribute name="count" type="xs:string" />
    <xs:attributeGroup ref="commonParmAttrs"/>
    <xs:attributeGroup ref="commonFieldAttrs"/>
  </xs:complexType>

  <xs:complexType name="unsignedIntElementType">
    <xs:simpleContent>
       <xs:extension base="xs:unsignedInt">
            <xs:attribute name="index" type="xs:nonNegativeInteger"  />
       </xs:extension>
    </xs:simpleContent>
  </xs:complexType>


<!-- Parametro integer breve (2 byte sul server) -->
   <xs:element name="shortParm" type="shortParmType" nillable="true" substitutionGroup="shortParmGroup"/>
      <xs:complexType name="shortParmType">
         <xs:simpleContent>
            <xs:extension base="shortFieldType">
              <xs:attributeGroup ref="commonParmAttrs"/>
            </xs:extension>
         </xs:simpleContent>
      </xs:complexType>

<!-- tipo schiera shortParm -->
  <xs:element name="arrayOfShortParm" type="shortParmArrayType" substitutionGroup="shortParmArrayGroup"
              nillable="true" />
  <xs:complexType name="shortParmArrayType">
    <xs:sequence>
      <xs:element name="i" type="shortElementType" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <!-- 'count' è necessario se si desidera immettere e/o emettere dati di schiera in formato XPCML. -->
    <xs:attribute name="count" type="xs:string" />
    <xs:attributeGroup ref="commonParmAttrs"/>
    <xs:attributeGroup ref="commonFieldAttrs"/>
  </xs:complexType>

  <xs:complexType name="shortElementType">
    <xs:simpleContent>
       <xs:extension base="xs:short">
            <xs:attribute name="index" type="xs:nonNegativeInteger"  />
       </xs:extension>
    </xs:simpleContent>
  </xs:complexType>


<!-- Parametro integer breve senza segno (2 byte sul server) -->
    <xs:element name="unsignedShortParm" type="unsignedShortParmType" nillable="true"
                substitutionGroup="unsignedShortParmGroup"   />
      <xs:complexType name="unsignedShortParmType">
         <xs:simpleContent>
            <xs:extension base="unsignedShortFieldType">
              <xs:attributeGroup ref="commonParmAttrs"/>
            </xs:extension>
         </xs:simpleContent>
      </xs:complexType>

<!-- tipo schiera unsignedShortParm  -->
  <xs:element name="arrayOfUnsignedShortParm" type="unsignedShortParmArrayType"
              substitutionGroup="unsignedShortParmArrayGroup" nillable="true" />
  <xs:complexType name="unsignedShortParmArrayType">
    <xs:sequence>
      <xs:element name="i" type="unsignedShortElementType" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <!-- 'count' è necessario se si desidera immettere e/o emettere dati di schiera in formato XPCML. -->
    <xs:attribute name="count" type="xs:string" />
    <xs:attributeGroup ref="commonParmAttrs"/>
```

```
    <xs:attributeGroup ref="commonFieldAttrs"/>
  </xs:complexType>

  <xs:complexType name="unsignedShortElementType">
    <xs:simpleContent>
       <xs:extension base="xs:unsignedShort">
            <xs:attribute name="index" type="xs:nonNegativeInteger"  />
       </xs:extension>
    </xs:simpleContent>
  </xs:complexType>


<!-- Parametro integer lungo (8 byte sul server) -->
    <xs:element name="longParm" type="longParmType" nillable="true" substitutionGroup="longParmGroup"  />
      <xs:complexType name="longParmType">
         <xs:simpleContent>
            <xs:extension base="longFieldType">
              <xs:attributeGroup ref="commonParmAttrs"/>
            </xs:extension>
         </xs:simpleContent>
      </xs:complexType>

<!-- tipo schiera longParm -->
  <xs:element name="arrayOfLongParm" type="longParmArrayType" substitutionGroup="longParmArrayGroup"
              nillable="true" />
  <xs:complexType name="longParmArrayType">
    <xs:sequence>
      <xs:element name="i" type="longElementType" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <!-- 'count' è necessario se si desidera immettere e/o emettere dati di schiera in formato XPCML. -->
    <xs:attribute name="count" type="xs:string" />
    <xs:attributeGroup ref="commonParmAttrs"/>
    <xs:attributeGroup ref="commonFieldAttrs"/>
  </xs:complexType>

  <xs:complexType name="longElementType">
    <xs:simpleContent>
       <xs:extension base="xs:long">
            <xs:attribute name="index" type="xs:nonNegativeInteger"  />
       </xs:extension>
    </xs:simpleContent>
  </xs:complexType>


<!-- Parametro ZonedDecimal -->
    <xs:element name="zonedDecimalParm" type="zonedDecimalParmType" nillable="true"
                substitutionGroup="zonedDecimalParmGroup"   />
      <xs:complexType name="zonedDecimalParmType">
         <xs:simpleContent>
            <xs:extension base="zonedDecimalFieldType">
              <xs:attributeGroup ref="commonParmAttrs"/>
            </xs:extension>
         </xs:simpleContent>
      </xs:complexType>

<!-- tipo schiera zonedDecimalParm -->
  <xs:element name="arrayOfZonedDecimalParm" type="zonedDecimalParmArrayType"
              substitutionGroup="zonedDecimalParmArrayGroup" nillable="true" />
  <xs:complexType name="zonedDecimalParmArrayType">
    <xs:sequence>
      <xs:element name="i" type="zonedDecimalElementType" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attributeGroup ref="commonParmAttrs"/>
    <xs:attributeGroup ref="commonFieldAttrs"/>
    <!-- 'count' è necessario se si desidera immettere e/o emettere dati di schiera in formato XPCML. -->
    <xs:attribute name="count" type="xs:string" />
    <!-- Il numero totale di cifre nel campo ('length' in PCML). -->
```

```
    <xs:attribute name="totalDigits" type="xs:positiveInteger"/>
    <!-- Il numero di cifre frazionarie ('precision' in PCML). -->
    <xs:attribute name="fractionDigits" type="xs:nonNegativeInteger"/>
  </xs:complexType>

  <xs:complexType name="zonedDecimalElementType">
    <xs:simpleContent>
       <xs:extension base="xs:decimal">
            <xs:attribute name="index" type="xs:nonNegativeInteger"  />
       </xs:extension>
    </xs:simpleContent>
  </xs:complexType>

<!-- Parametro packedDecimal -->
    <xs:element name="packedDecimalParm" type="packedDecimalParmType" nillable="true"
                substitutionGroup="packedDecimalParmGroup"   />
      <xs:complexType name="packedDecimalParmType">
         <xs:simpleContent>
            <xs:extension base="packedDecimalFieldType">
              <xs:attributeGroup ref="commonParmAttrs"/>
            </xs:extension>
         </xs:simpleContent>
      </xs:complexType>

<!-- tipo di schiera packedDecimalParm -->
  <xs:element name="arrayOfPackedDecimalParm" type="packedDecimalParmArrayType"
              substitutionGroup="packedDecimalParmArrayGroup" nillable="true" />
  <xs:complexType name="packedDecimalParmArrayType">
    <xs:sequence>
      <xs:element name="i" type="packedDecimalElementType" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attributeGroup ref="commonParmAttrs"/>
    <xs:attributeGroup ref="commonFieldAttrs"/>
    <!-- 'count' è necessario se si desidera immettere e/o emettere dati di schiera in formato XPCML. -->
    <xs:attribute name="count" type="xs:string" />
    <xs:attribute name="totalDigits" type="xs:positiveInteger"/>
    <xs:attribute name="fractionDigits" type="xs:nonNegativeInteger"/>
  </xs:complexType>

  <xs:complexType name="packedDecimalElementType">
    <xs:simpleContent>
       <xs:extension base="xs:decimal">
            <xs:attribute name="index" type="xs:nonNegativeInteger"  />
       </xs:extension>
    </xs:simpleContent>
  </xs:complexType>

<!-- Parametro Float (4 byte sul server) -->
   <xs:element name="floatParm" type="floatParmType" nillable="true" substitutionGroup="floatParmGroup"/>
      <xs:complexType name="floatParmType">
         <xs:simpleContent>
            <xs:extension base="floatFieldType">
              <xs:attributeGroup ref="commonParmAttrs"/>
            </xs:extension>
         </xs:simpleContent>
      </xs:complexType>

<!-- tipo di schiera floatParm -->
  <xs:element name="arrayOfFloatParm" type="floatParmArrayType" substitutionGroup="floatParmArrayGroup"
              nillable="true" />
  <xs:complexType name="floatParmArrayType">
    <xs:sequence>
      <xs:element name="i" type="floatElementType" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <!-- 'count' è necessario se si desidera immettere e/o emettere dati di schiera in formato XPCML. -->
    <xs:attribute name="count" type="xs:string" />
    <xs:attributeGroup ref="commonParmAttrs"/>
```

```
    <xs:attributeGroup ref="commonFieldAttrs"/>
  </xs:complexType>

  <xs:complexType name="floatElementType">
    <xs:simpleContent>
       <xs:extension base="xs:float">
            <xs:attribute name="index" type="xs:nonNegativeInteger"  />
       </xs:extension>
    </xs:simpleContent>
  </xs:complexType>


<!-- Parametro Double (8 bytes sul server) -->
    <xs:element name="doubleParm" type="doubleParmType" nillable="true"
                substitutionGroup="doubleParmGroup"   />
      <xs:complexType name="doubleParmType">
         <xs:simpleContent>
            <xs:extension base="doubleFieldType">
              <xs:attributeGroup ref="commonParmAttrs"/>
            </xs:extension>
         </xs:simpleContent>
      </xs:complexType>

<!-- tipo schiera doubleParm -->
  <xs:element name="arrayOfDoubleParm" type="doubleParmArrayType"
              substitutionGroup="doubleParmArrayGroup" nillable="true" />
  <xs:complexType name="doubleParmArrayType">
    <xs:sequence>
      <xs:element name="i" type="doubleElementType" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <!-- 'count' è necessario se si desidera immettere e/o emettere dati di schiera in formato XPCML. -->
    <xs:attribute name="count" type="xs:string" />
    <xs:attributeGroup ref="commonParmAttrs"/>
    <xs:attributeGroup ref="commonFieldAttrs"/>
  </xs:complexType>

  <xs:complexType name="doubleElementType">
    <xs:simpleContent>
       <xs:extension base="xs:double">
            <xs:attribute name="index" type="xs:nonNegativeInteger"  />
       </xs:extension>
    </xs:simpleContent>
  </xs:complexType>


<!-- Parametro binario Hex (qualsiasi numero di byte; senza segno) -->
    <xs:element name="hexBinaryParm" type="hexBinaryParmType"  substitutionGroup="hexBinaryParmGroup"  />
      <xs:complexType name="hexBinaryParmType">
         <xs:simpleContent>
            <xs:extension base="hexBinaryFieldType">
              <!-- La lunghezza del campo in byte ('length' in PCML). -->
              <xs:attribute name="totalBytes" type="xs:string"/>
              <xs:attributeGroup ref="commonParmAttrs"/>
            </xs:extension>
         </xs:simpleContent>
      </xs:complexType>

<!-- tipo schiera hexBinaryParm -->
  <xs:element name="arrayOfHexBinaryParm" type="hexBinaryParmArrayType"
              substitutionGroup="hexBinaryParmArrayGroup" nillable="true" />
  <xs:complexType name="hexBinaryParmArrayType">
    <xs:sequence>
      <xs:element name="i" type="hexBinaryElementType" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="totalBytes" type="xs:string"/>
    <!-- 'count' è necessario se si desidera immettere e/o emettere dati di schiera in formato XPCML. -->
    <xs:attribute name="count" type="xs:string" />
```

```
    <xs:attributeGroup ref="commonParmAttrs"/>
    <xs:attributeGroup ref="commonFieldAttrs"/>
  </xs:complexType>

  <xs:complexType name="hexBinaryElementType">
    <xs:simpleContent>
       <xs:extension base="xs:hexBinary">
            <xs:attribute name="index" type="xs:nonNegativeInteger"  />
       </xs:extension>
    </xs:simpleContent>
  </xs:complexType>

<!-- tipo Structure parm -->
  <xs:element name="structParm" type="structParmType"  substitutionGroup="structParmGroup"   />
      <xs:complexType name="structParmType">
         <xs:complexContent>
            <xs:extension base="structureParmArray">
              <xs:attribute name="struct" type="string50"/>
              <!-- Specifica se il parametro viene passato per valore o riferimento ('passby' in PCML).-->
              <!-- Valore consentito solo per parametri integer. -->
              <xs:attribute name="passMode" type="passModeType"/>
              <!-- 'count' è necessario se si desidera immettere e/o emettere dati di schiera in formato XPCML. -->
              <xs:attribute name="count" type="xs:string"/>
            </xs:extension>
         </xs:complexContent>
      </xs:complexType>

<!-- tipo schiera parametro Structure -->
  <xs:element name="arrayOfStructParm" type="structParmArrayType"
              substitutionGroup="structParmArrayGroup" nillable="true" />
  <xs:complexType name="structParmArrayType">
    <xs:sequence>
        <!-- la tag struct_i rappresenta gli elementi di schiera struct o struct parm. -->
        <xs:element name="struct_i" type="structElementType" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <!-- 'count' è necessario se si desidera immettere e/o emettere dati di schiera in formato XPCML. -->
    <xs:attribute name="count" type="xs:string" />
    <xs:attributeGroup ref="commonParmAttrs"/>
    <xs:attributeGroup ref="commonFieldAttrs"/>
    <xs:attribute name="struct" type="string50"/>
  </xs:complexType>

  <xs:complexType name="structElementType">
    <xs:complexContent>
       <xs:extension base="structureParmArray">
            <xs:attribute name="index" type="xs:nonNegativeInteger"  />
       </xs:extension>
    </xs:complexContent>
  </xs:complexType>


<!-- Elemento struct -->
  <xs:element name="struct" type="structureParmArray" substitutionGroup="structOrProgram" />

<!-- tipo schiera struct -->
  <xs:element name="arrayOfStruct" type="structArrayType" substitutionGroup="structArrayGroup"
              nillable="true" />
  <xs:complexType name="structArrayType">
    <xs:sequence>
       <!-- La tag struct_i  rappresenta elementi struct in una schiera. -->
       <xs:element name="struct_i" type="structElementType" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <!-- Il nome di struct. -->
    <xs:attribute name="name" type="string50"/>
    <!-- Numero di elementi nella schiera. -->
    <!-- 'count' è necessario se si desidera immettere e/o emettere dati di schiera in formato XPCML. -->
    <xs:attribute name="count" type="xs:string" />
```

```
    <!-- Specifica se questa è una struct di immissione, emissione o immissione-emissione ('usage' in PCML). -->
    <xs:attribute name="passDirection" type="passDirectionType"/>
    <!-- Lo scostamento rispetto a struct in un parametro di emissione. -->
    <xs:attribute name="offset" type="xs:string" />
    <!-- L'ubicazione di base a cui di riferisce l'attributo 'offset'. -->
    <xs:attribute name="offsetFrom" type="xs:string" />
    <!-- Il numero di byte da riservare per i dati di emissione per l'elemento. -->
    <xs:attribute name="outputSize" type="xs:string" />
    <!-- La versione più recente di OS/400 nella quale esiste questo elemento. -->
    <xs:attribute name="minvrm" type="string10" />
    <!-- La versione meno recente di OS/400 nella quale esiste questo elemento. -->
    <xs:attribute name="maxvrm" type="string10" />
  </xs:complexType>


<!-- Attributi comuni a tutti i tipi di campi dati. -->
  <xs:attributeGroup name="commonParmAttrs">
     <!-- Specifica se questo è un parametro di immissione, emissione o immissione-emissione ('usage' in PCML). -->
     <!-- Il valore predefinito se non ne specificato alcuno è 'inherit'. -->
     <xs:attribute name="passDirection" type="passDirectionType"/>
     <!-- Specifica se il parametro viene passato per riferimento o valore ('passby' in PCML). -->
     <!-- Il valore predefinito se non ne è specificato alcuno è 'reference'. -->
     <xs:attribute name="passMode" type="passModeType" />
     <!-- Lo scostamento rispetto all'elemento entro un parametro di emissione. -->
     <!-- Il valore predefinito se non ne è specificato alcuno è 0. -->
     <xs:attribute name="offset" type="xs:string" />
     <!-- L'ubicazione di base a cui di riferisce l'attributo 'offset'. -->
     <xs:attribute name="offsetFrom" type="xs:string" />
     <!-- Il numero di byte da riservare per i dati di emissione per l'elemento. -->
     <xs:attribute name="outputSize" type="xs:string" />
     <!-- La versione più recente di OS/400 a cui si applica questo campo. -->
     <!-- Se non specificata, si presume che questo campo si applichi a tutte le versioni. -->
     <xs:attribute name="minvrm" type="string10" />
     <!-- La versione meno recente di OS/400 a cui si applica questo campo.  -->
     <!-- Se non specificata, si presume che questo campo si applichi a tutte le versioni. -->
     <xs:attribute name="maxvrm" type="string10" />
  </xs:attributeGroup>

  <xs:simpleType name="passDirectionType">
    <xs:restriction base="xs:string">
       <xs:enumeration value="in"/>
       <xs:enumeration value="inout"/>
       <xs:enumeration value="out"/>
       <xs:enumeration value="inherit"/>
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name="passModeType">
    <xs:restriction base="xs:string">
       <xs:enumeration value="value"/>
       <xs:enumeration value="reference"/>
     </xs:restriction>
  </xs:simpleType>

  <!-- I seguenti tipi servono a mantenere la compatibilità con PCML -->
  <xs:simpleType name="bidiStringTypeType">
    <xs:restriction base="xs:string">
       <xs:enumeration value="ST4"/>
       <xs:enumeration value="ST5"/>
       <xs:enumeration value="ST6"/>
       <xs:enumeration value="ST7"/>
       <xs:enumeration value="ST8"/>
       <xs:enumeration value="ST9"/>
       <xs:enumeration value="ST10"/>
       <xs:enumeration value="ST11"/>
       <xs:enumeration value="DEFAULT"/>
    </xs:restriction>
```

```
  </xs:simpleType>

  <xs:simpleType name="charType">
    <xs:restriction base="xs:string">
       <xs:enumeration value="onebyte"/>
       <xs:enumeration value="twobyte"/>
     </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name="trimType">
    <xs:restriction base="xs:string">
       <xs:enumeration value="none"/>
       <xs:enumeration value="left"/>
       <xs:enumeration value="right"/>
       <xs:enumeration value="both"/>
     </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name="returnValueType">
    <xs:restriction base="xs:string">
       <xs:enumeration value="void"/>
       <xs:enumeration value="integer"/>
     </xs:restriction>
  </xs:simpleType>


  <xs:complexType name="structureParmArray">
    <xs:sequence>
       <xs:group ref="structureParm" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="name" type="string50"/>
    <xs:attribute name="passDirection" type="passDirectionType"/>
    <xs:attribute name="offset" type="xs:string" />
    <xs:attribute name="offsetFrom" type="xs:string" />
    <xs:attribute name="outputSize" type="xs:string" />
    <xs:attribute name="minvrm" type="string10" />
    <xs:attribute name="maxvrm" type="string10" />
  </xs:complexType>


<!-- Uno structureParm è esattamente uno dei seguenti: stringParm, intParm,
     shortParm, longParm, zonedDecimalParm, packedDecimalParm, floatParm,
     doubleParm, or hexBinaryParm. -->
      <xs:group name="structureParm">
        <xs:choice>
          <xs:element ref="stringParmGroup"  />
          <xs:element ref="stringParmArrayGroup"  />
          <xs:element ref="intParmGroup"   />
          <xs:element ref="intParmArrayGroup"   />
          <xs:element ref="unsignedIntParmGroup"   />
          <xs:element ref="unsignedIntParmArrayGroup"   />
          <xs:element ref="shortParmGroup" />
          <xs:element ref="shortParmArrayGroup" />
          <xs:element ref="unsignedShortParmGroup"   />
          <xs:element ref="unsignedShortParmArrayGroup"   />
          <xs:element ref="longParmGroup"  />
          <xs:element ref="longParmArrayGroup"  />
          <xs:element ref="zonedDecimalParmGroup"  />
          <xs:element ref="zonedDecimalParmArrayGroup"  />
          <xs:element ref="packedDecimalParmGroup" />
          <xs:element ref="packedDecimalParmArrayGroup" />
          <xs:element ref="floatParmGroup" />
          <xs:element ref="floatParmArrayGroup" />
          <xs:element ref="doubleParmGroup" />
          <xs:element ref="doubleParmArrayGroup" />
          <xs:element ref="hexBinaryParmGroup" />
          <xs:element ref="hexBinaryParmArrayGroup" />
```

```
          <xs:element ref="structParmGroup" />
          <xs:element ref="structParmArrayGroup"/>
          <xs:element ref="structArrayGroup"/>
          <xs:element ref="struct"/>
        </xs:choice>
      </xs:group>


<!-- Schema di definizione campo. -->

  <!-- Definire i tipi di dati nativi iSeries di base. -->

     <xs:complexType name="zonedDecimal">
        <xs:simpleContent>
           <xs:extension base="xs:decimal">
                <xs:attribute name="totalDigits" type="xs:positiveInteger"  />
                <xs:attribute name="fractionDigits" type="xs:nonNegativeInteger" />
           </xs:extension>
        </xs:simpleContent>
     </xs:complexType>


     <xs:complexType name="packedDecimal">
        <xs:simpleContent>
           <xs:extension base="xs:decimal">
                <xs:attribute name="totalDigits" type="xs:positiveInteger" />
                <xs:attribute name="fractionDigits" type="xs:nonNegativeInteger" />
           </xs:extension>
        </xs:simpleContent>
     </xs:complexType>


    <xs:complexType name="structureFieldArray">
     <xs:sequence>
        <xs:group ref="structureField" minOccurs="1" maxOccurs="unbounded"/>
     </xs:sequence>
     <xs:attribute name="name" type="string50"/>
     <!-- 'count' è necessario se si desidera immettere e/o emettere dati di schiera in formato XPCML. -->
     <xs:attribute name="count" type="xs:string"/>
  </xs:complexType>



  <!-- Tipo astratto per "struct o program". -->
  <xs:element name="structOrProgram" abstract="true" />


  <!-- Tipo astratto per tutti i tipi di campi dati. -->
  <xs:element name="stringFieldGroup" type="stringFieldType" abstract="true" />
  <xs:element name="intFieldGroup" type="intFieldType" abstract="true" />
  <xs:element name="unsignedIntFieldGroup" type="unsignedIntFieldType" abstract="true" />
  <xs:element name="shortFieldGroup" type="shortFieldType" abstract="true" />
  <xs:element name="unsignedShortFieldGroup" type="unsignedShortFieldType" abstract="true" />
  <xs:element name="longFieldGroup" type="longFieldType" abstract="true" />
  <xs:element name="zonedDecimalFieldGroup" type="zonedDecimalFieldType" abstract="true" />
  <xs:element name="packedDecimalFieldGroup" type="packedDecimalFieldType" abstract="true" />
  <xs:element name="floatFieldGroup" type="floatFieldType" abstract="true" />
  <xs:element name="doubleFieldGroup" type="doubleFieldType" abstract="true" />
  <xs:element name="hexBinaryFieldGroup" type="hexBinaryFieldType" abstract="true" />
  <xs:element name="structFieldGroup" type="structFieldType" abstract="true" />



  <!-- Dichiarare ogni elemento campo in modo che sia di un tipo campo specifico. -->
  <xs:element name="stringField" type="stringFieldType" substitutionGroup="stringFieldGroup"
              nillable="true"/>
  <xs:element name="intField" type="intFieldType" nillable="true"
```

```
              substitutionGroup="intFieldGroup" />
  <xs:element name="unsignedIntField" type="unsignedIntFieldType"
              substitutionGroup="unsignedIntFieldGroup" nillable="true"/>
  <xs:element name="shortField" type="shortFieldType" nillable="true"
              substitutionGroup="shortFieldGroup" />
  <xs:element name="unsignedShortField" type="unsignedShortFieldType" nillable="true"
              substitutionGroup="unsignedShortFieldGroup"  />
  <xs:element name="longField" type="longFieldType" nillable="true"
              substitutionGroup="longFieldGroup" />
  <xs:element name="hexBinaryField" type="hexBinaryFieldType" nillable="true"
              substitutionGroup="hexBinaryFieldGroup" />
  <xs:element name="zonedDecimalField" type="zonedDecimalFieldType" nillable="true"
              substitutionGroup="zonedDecimalFieldGroup"  />
  <xs:element name="packedDecimalField" type="packedDecimalFieldType" nillable="true"
              substitutionGroup="packedDecimalFieldGroup" />
  <xs:element name="doubleField" type="doubleFieldType" nillable="true"
              substitutionGroup="doubleFieldGroup" />
  <xs:element name="floatField" type="floatFieldType" nillable="true"
              substitutionGroup="floatFieldGroup" />


  <xs:element name="structField" type="structFieldType" nillable="true"
              substitutionGroup="structFieldGroup" />

<!-- Uno StructureField è esattamente uno dei seguenti: stringField, intField,
     shortField, longField, zonedDecimalField, packedDecimalField, floatField,
     doubleField, or hexBinaryField. -->
      <xs:group name="structureField">
        <xs:choice>
          <xs:element ref="stringFieldGroup"/>
          <xs:element ref="intFieldGroup"/>
          <xs:element ref="unsignedIntFieldGroup"/>
          <xs:element ref="shortFieldGroup"/>
          <xs:element ref="unsignedShortFieldGroup"/>
          <xs:element ref="longFieldGroup"/>
          <xs:element ref="zonedDecimalFieldGroup"/>
          <xs:element ref="packedDecimalFieldGroup"/>
          <xs:element ref="floatFieldGroup"/>
          <xs:element ref="doubleFieldGroup"/>
          <xs:element ref="hexBinaryFieldGroup"/>
          <xs:element ref="structParmGroup"/>
          <xs:element ref="struct"/>
        </xs:choice>
      </xs:group>

  <!-- Campo carattere -->
  <!-- Corrisponde a AS400Text. -->
   <xs:complexType name="stringFieldType">
     <xs:simpleContent>
        <xs:extension base="xs:string">
          <!-- Numero di caratteri. -->
          <xs:attribute name="length" type="xs:string"/>
          <!-- Indica la codifica del campo (CCSID) sul server. -->
          <xs:attribute name="ccsid" type="xs:string"/>
          <xs:attribute name="trim" type="trimType" />
          <xs:attribute name="bytesPerChar" type="charType" />
          <xs:attribute name="bidiStringType" type="bidiStringTypeType" />
          <xs:attributeGroup ref="commonFieldAttrs"/>
        </xs:extension>
     </xs:simpleContent>
   </xs:complexType>


  <!-- Campo hexBinary -->
  <!-- Corrisponde a AS400ByteArray. -->
   <xs:complexType name="hexBinaryFieldType">
     <xs:simpleContent>
```

```
      <xs:extension base="xs:hexBinary">
        <xs:attributeGroup ref="commonFieldAttrs"/>
      </xs:extension>
    </xs:simpleContent>
 </xs:complexType>


<!-- Campo Float -->
<!-- Corrisponde a AS400Float4. -->
 <xs:complexType name="floatFieldType">
   <xs:simpleContent>
       <xs:extension base="xs:float">
           <xs:attributeGroup ref="commonFieldAttrs"/>
       </xs:extension>
   </xs:simpleContent>
  </xs:complexType>

<!-- Campo zonedDecimal -->
<!-- Corrisponde a AS400ZonedDecimal. -->
 <xs:complexType name="zonedDecimalFieldType">
   <xs:simpleContent>
       <xs:extension base="zonedDecimal">
           <xs:attributeGroup ref="commonFieldAttrs"/>
       </xs:extension>
   </xs:simpleContent>
 </xs:complexType>

<!-- Campo packedDecimal -->
<!-- Corrisponde a AS400PackedDecimal. -->
 <xs:complexType name="packedDecimalFieldType">
   <xs:simpleContent>
       <!-- In DDS, i valori "binary" sono composti da 1-18 cifre; se la lunghezza del campo è
            maggiore di 9, allora il valore delle posizioni decimali debve essere 0. -->
       <xs:extension base="packedDecimal">
           <xs:attributeGroup ref="commonFieldAttrs"/>
       </xs:extension>
   </xs:simpleContent>
 </xs:complexType>

<!-- Campo int -->
<!-- Corrisponde a AS400Bin4. -->
 <xs:complexType name="intFieldType">
   <xs:simpleContent>
       <xs:extension base="xs:int">
          <xs:attributeGroup ref="commonFieldAttrs"/>
       </xs:extension>
   </xs:simpleContent>
 </xs:complexType>

<!-- Campo int senza segno -->
<!-- Corrisponde a AS400Bin4. -->
 <xs:complexType name="unsignedIntFieldType">
   <xs:simpleContent>
       <xs:extension base="xs:unsignedInt">
          <xs:attributeGroup ref="commonFieldAttrs"/>
       </xs:extension>
   </xs:simpleContent>
 </xs:complexType>

<!-- Campo short -->
<!-- Corrisponde a AS400Bin2. -->
 <xs:complexType name="shortFieldType">
   <xs:simpleContent>
       <xs:extension base="xs:short">
          <xs:attributeGroup ref="commonFieldAttrs"/>
       </xs:extension>
   </xs:simpleContent>
```

```
   </xs:complexType>

  <!-- Campo short senza segno -->
  <!-- Corrisponde a AS400Bin2. -->
   <xs:complexType name="unsignedShortFieldType">
     <xs:simpleContent>
         <xs:extension base="xs:unsignedShort">
            <xs:attributeGroup ref="commonFieldAttrs"/>
         </xs:extension>
     </xs:simpleContent>
   </xs:complexType>


  <!-- Campo long -->
  <!-- Corrisponde a AS400Bin8. -->
   <xs:complexType name="longFieldType">
     <xs:simpleContent>
         <xs:extension base="xs:long">
            <xs:attributeGroup ref="commonFieldAttrs"/>
         </xs:extension>
     </xs:simpleContent>
   </xs:complexType>

  <!-- Campo double -->
  <!-- Corrisponde a AS400Float8. -->
   <xs:complexType name="doubleFieldType">
     <xs:simpleContent>
         <xs:extension base="xs:double">
            <xs:attributeGroup ref="commonFieldAttrs"/>
         </xs:extension>
     </xs:simpleContent>
   </xs:complexType>



  <!-- Campo struct -->
   <xs:complexType name="structFieldType">
     <xs:simpleContent>
       <xs:extension base="xs:string">
         <xs:attribute name="struct" type="string50"/>
         <xs:attributeGroup ref="commonFieldAttrs"/>
       </xs:extension>
     </xs:simpleContent>
   </xs:complexType>


<!-- Attributi comuni a tutti i tipi di campi dati. -->
<xs:attributeGroup name="commonFieldAttrs">
   <xs:attribute name="name" type="string50"/>
   <xs:attribute name="columnHeading1" type="string20"/>
   <xs:attribute name="columnHeading2" type="string20"/>
   <xs:attribute name="columnHeading3" type="string20"/>
   <xs:attribute name="description" type="string50"/>
   <xs:attribute name="defaultValue" type="xs:string"/><!-- Lunghezza massima di stringa 65535 caratteri. -->
   <xs:attribute name="nullable" type="xs:boolean"/>
   <xs:attribute name="isEmptyString" type="xs:boolean"/><!-- Indicare che si tratta di una stringa vuota. -->
</xs:attributeGroup>

<!-- Tipi di utilità.  -->

<xs:simpleType name="ccsidType">
                     <xs:restriction base="xs:nonNegativeInteger">
                      <xs:maxInclusive value="65535"/>
                     </xs:restriction>
                    </xs:simpleType>

<xs:simpleType name="string10">
```

```
                    <xs:restriction base="xs:string">
                     <xs:maxLength value="10"/>
                    </xs:restriction>
                   </xs:simpleType>

<xs:simpleType name="string20">
                    <xs:restriction base="xs:string">
                     <xs:maxLength value="20"/>
                    </xs:restriction>
                   </xs:simpleType>

<xs:simpleType name="string50">
                    <xs:restriction base="xs:string">
                     <xs:maxLength value="50"/>
                    </xs:restriction>
                   </xs:simpleType>
</xs:schema>
```

Sintassi XPCML:

Lo schema XPCML definisce diverse tag di elemento ed ogni tag di elemento contiene tag di attributo. La seguente tabella elenca gli elementi differenti che è possibile dichiarare e definire nei file sorgente XPCML. Ogni voce nella prima colonna si collega alla sezione appropriata dello schema XPCML.

Tag XPCML	Descrizione	Tag PCML equivalente
doubleParm	Definisce un parametro double	data (type=float, length=8)
arrayOfDoubleParm	Definisce un parametro che è una schiera di double	
floatParm	Definisce un parametro float	data (type=float, length=4)
arrayOfFloatParm	Definisce un parametro che è una schiera di float	
hexBinaryParm	Definisce un parametro byte rappresentato in hex	byte (equivalente grezzo, rappresentato in hex)
arrayOfHexBinaryParm	Definisce un parametro che è una schiera di hexBinaries	
intParm	Definisce un parametro integer	dati (type=int, length=4)
arrayOfIntParm	Definisce un parametro che è una schiera di integer	
longParm	Definisce un parametro long	dati (type=int, length=8)
arrayOfLongParm	Definisce un parametro che è una schiera di long	
packedDecimalParm	Definisce un parametro packed decimal	dati (type=packed)
arrayOfPackedDecimalParm	Definisce un parametro che è una schiera di packed decimal	
parameterList	Segnala che la tag di inclusione rappresenta tutte le definizioni di parametro per il programma	
program	Inizia e termina l'XML che descrive una chiamata al programma	program
shortParm	Definisce un parametro short	dati (type int, lenght 2)
arrayOfShortParm	Definisce un parametro che è una schiera di short	
stringParm	Definisce un parametro string	

Tag XPCML	Descrizione	Tag PCML equivalente
arrayOfStringParm	Definisce un parametro che è una schiera di string	
struct	Definisce una struttura denominata che è possibile specificare come un argomento per un programma o come un campo entro un'altra struttura denominata	struct
arrayOfStruct	Definisce una schiera di struct	
structParm	Rappresenta un riferimento ad una tag struct trovata altrove nel documento XPCML che si desidera includere in una specifica ubicazione nel documento	dati (type=struct)
arrayOfStructParm	Definisce un parametro che è una schiera di parametri struct	
unsignedIntParm	Definisce un parametro integer senza segno	dati (type=int, length=4, precision=32)
arrayOfUnsignedIntParm	Definisce un parametro che è una schiera di integer senza segno	
unsignedShortParm	Definisce un parametro short senza segno	dati (type=int, length=2, precision=16)
arrayOfUnsignedShortParm	Definisce un parametro che è una schiera di short senza segno	
xpcml	Inizia e termina il file sorgente XPCML che descrive il formato della chiamata al programma	
zonedDecimalParm	Definisce un parametro zoned decimal	dati (type zoned)
arrayOfZonedDecimalParm	Definisce un parametro che è una schiera di zoned decimal	

Attributi tag XPCML:

Lo schema XPCML definisce diverse tag di elemento ed ogni tag di elemento contiene tag di attributo. La seguente tabella elenca e descrive i differenti attributi per ogni elemento.

Per informazioni più specifiche e dettagliate sulle tag XPCML ed i relativi attributi, consultare Schema XPCML.

Tag XPCML	Attributo	Descrizione
hexBinaryParm	riempire le ultime 2 colonne con dati e decidere il formato	
arrayOfHexBinaryParm		
doubleParm	Definisce un parametro double	float (lenght 8)
arrayOfDoubleParm	Definisce un parametro che è una schiera di double	
floatParm	Definisce un parametro float	dati (type float, lenght 4)
arrayOfFloatParm	Definisce un parametro che è una schiera di float	

Tag XPCML	Attributo	Descrizione
intParm	Definisce un parametro integer	dati (type int, lenght 4)
arrayOfIntParm	Definisce un parametro che è una schiera di integer	
longParm	Definisce un parametro long	dati (type int, lenght 8)
arrayOfLongParm	Definisce un parametro che è una schiera di long	
packedDecimalParm	Definisce un parametro packed decimal	dati (type packed)
arrayOfPackedDecimalParm	Definisce un parametro che è una schiera di packed decimal	
parameterList	Segnala che la tag di inclusione rappresenta tutte le definizioni di parametro per il programma	
program	Inizia e termina l'XML che descrive una chiamata al programma	
shortParm	Definisce un parametro short	dati (type int, lenght 2)
arrayOfShortParm	Definisce un parametro che è una schiera di short	
stringParm	Definisce un parametro string	
arrayOfStringParm	Definisce un parametro che è una schiera di string	
struct	Definisce una struttura denominata che è possibile specificare come un argomento per un programma o come un campo entro un'altra struttura denominata	
arrayOfStruct	Definisce una schiera di struct	
structParm	Rappresenta un riferimento ad una tag struct trovata altrove nel documento XPCML che si desidera includere in una specifica ubicazione nel documento	dati (type struct)
arrayOfStructParm	Definisce un parametro che è una schiera di parametri struct	
unsignedIntParm	Definisce un parametro integer senza segno	dati (type int, length 4, precision 32)
arrayOfUnsignedIntParm	Definisce un parametro che è una schiera di integer senza segno	
unsignedShortParm	Definisce un parametro short senza segno	data (type int, length 2, precision 16)
arrayOfUnsignedShortParm	Definisce un parametro che è una schiera di short senza segno	
xpcml	Inizia e termina il file sorgente XPCML che descrive il formato della chiamata al programma	
zonedDecimalParm	Definisce un parametro zoned decimal	dati (type zoned)
arrayOfZonedDecimalParm	Definisce un parametro che è una schiera di zoned decimal	

Utilizzo di XPCML

L'utilizzo di XPCML è simile all'utilizzo di PCML. Le seguenti istruzioni servono come da direttive generali per le operazioni che è necessario eseguire per utilizzare XPCML:

1. Utilizzare XPCML per descrivere le specifiche per la chiamata al programma
2. Creare un oggetto ProgramCallDocument
3. Utilizzare ProgramCallDocument.callProgram() per eseguire il programma

Nonostante le similitudini rispetto all'utilizzo di PCML, l'utilizzo di XPCML offre una funzionalità potenziata:

- Fare sì che il programma di analisi convalidi automaticamente i valori di parametro
- Specificare e passare valori per parametri di programma
- Richiamare i risultati di una chiamata di programma nel server iSeries in XPCML
- Trasformare un documento PCML esistente nell'equivalente documento XPCML
- Estendere lo schema XPCML in modo da definire elementi ed attributi nuovi semplici e complessi

Ad esempio, IBM Toolbox per Java consente di estendere lo schema XPCML per creare nuovi parametri e tipi di dati. E' possibile utilizzare questa capacità XPCML condensare i file sorgente XPCML, il che rende i file più facili da leggere ed il codice più facile da utilizzare.

Per ulteriori informazioni sull'utilizzo di XPCML, consultare le seguenti pagine:

Conversione di PCML in XPCML
Acquisire informazioni su come trasformare i documenti PCML esistenti in documenti XPCML equivalenti.

Utilizzo di XPCML per chiamare un programma sul server iSeries
Acquisire informazioni su come creare un oggetto ProgramCallDocument che utilizza XPCML per chiamare un programma sul server.

Come ottenere i risultati della chiamata al programma come XPCML
Acquisire informazioni su come richiamare i risultati della chiamata ad un programma server come XPCML.

Inoltro dei valori di parametro come XPCML
Acquisire informazioni su come impostare valori per i parametri di programma in XPCML e passare quei valori. Esaminare differenti tecniche per definire valori di parametro costanti e schiere di dati.

Utilizzo di XPCML condensato
Acquisire informazioni su come il condensare un documento XPCML ne rende più facile la lettura e l'utilizzo. Acquisire informazioni sull'utilizzo di XPCML condensato per creare un oggetto ProgramCallDocument e ottenere i risultati della chiamata al programma come XPCML condensato.

Identificazione degli errori di analisi in XPCML
Acquisire informazioni su come identificare e registrare avvertenze utili ed errori di analisi non irreversibili che a volte si presentano durante l'analisi di un documento XPCML.

Conversione del PCML esistente in XPCML:

La classe ProgramCallDocument contiene il metodo transformPCMLToXPCML che consente all'utente di trasformare i documenti PCML esistenti in documenti XPCML equivalenti.

XPCML ha definizioni confrontabili per tutti gli elementi e gli attributi che è possibile definire in PCML. Utilizzando transformPCMLToXPCML() si converte la rappresentazione PCML degli elementi e attributi nell'XPCML equivalente.

Si noti che in alcuni casi, gli attributi XPCML equivalenti hanno un nome differente rispetto a PCML. Ad esempio, l'attributo ″usage″ in PCML è l'attributo ″passDirection″ in XPCML. Per ulteriori informazioni su come utilizzare XPCML in confronto a PCML, consultare Schema e sintassi XPCML.

Il metodo prende il documento PCML esistente, che l'utente ha passato come oggetto InputStream e crea l'XPCML equivalente come oggetto OutputStream. Poiché transformPCMLToXPCML() è un metodo statico, è possibile chiamarlo senza prima creare un oggetto ProgramCallDocument.

Esempio: conversione di un documento PCML in un documento XPCML

Il seguente esempio mostra come convertire un documento PCML (denominato myPCML.pcml) in un documento XPCML (denominato myXPCML.xpcml).

Nota: è necessario specificare .xpcml come estensione file per i file XPCML. L'utilizzo di .xpcml come estensione del file assicura che la classe ProgramCallDocument riconosca il file come XPCML. Se non si specifica un'estensione, ProgramCallDocument presuppone che il file sia PCML.

Documento PCML myPCML.pcml

```
<!-- myPCML.pcml -->
<pcml version="4.0">
   <program name="prog1" path="/QSYS.LIB/W95LIB.LIB/PROG1.PGM">
      <data type="char" name="parm1" usage="in" passby="reference"
            minvrm="V5R2M0" ccsid="37" length="10" init="Value 1"/>
   </program>
</pcml>
```

Codice Java per convertire myPCML.pcml in myPCML.xpcml

```
            try {
      InputStream pcmlStream = new FileInputStream("myPCML.pcml");
      OutputStream xpcmlStream = new FileOutputStream("myXPCML.xpcml");
      ProgramCallDocument.transformPCMLToXPCML(pcmlStream, xpcmlStream);
    }
            catch (Exception e) {
      System.out.println("error:  - "+e.getMessage());
              e.printStackTrace();
    }
```

Documento XPCML risultante myXPCML.xpcml

```
<?xml version="1.0" encoding="UTF-8"?>
   <!-- myXPCML.xpcml -->
   <xpcml xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:noNamespaceSchemaLocation="xpcml.xsd" version="4.0">
   <program name="prog1" path="/QSYS.LIB/W95LIB.LIB/PROG1.PGM">
      <parameterList>
         <stringParm name="parm1" passDirection="in" passMode="reference"
                     minvrm="V5R2M0" ccsid="37" length="10">Value 1</stringParm>
</parameterList>
       </program>
</xpcml>
```

Per ulteriori informazioni su transformPCMLToXPCML() e sulla classe ProgramCallDocument, consultare la seguente pagina:

Informazioni javadoc ProgramCallDocument

Utilizzo di XPCML per chiamare un programma sul server iSeries:

Dopo aver creato il file XPCML, è necessario creare un oggetto ProgramCallDocument che può utilizzare specifiche e valori dati XPCML per chiamare un programma sul server iSeries. Creare ProgramCallDocument XPCML passando il nome del file XPCML nel programma di creazione

ProgramCallDocument. Creando ProgramCallDocument XPCML in questo modo prima si analizza e si convalida il documento XPCML, quindi si crea l'oggetto ProgramCallDocument.

Per analizzare e convalidare il documento XPCML, accertarsi che CLASSPATH includa un programma di analisi XML a convalida completa. Per ulteriori informazioni sui requisiti per eseguire XPCML, consultare la seguente pagina:

"Requisiti per l'utilizzo di XPCML" a pagina 420

Il seguente esempio mostra come creare un oggetto ProgramCallDocument per il file XPCML, myXPCML.xpcml.

```
system = new AS400();
// Creare ProgramCallDocument in cui analizzare il file.
ProgramCallDocument xpcmlDoc =
   new ProgramCallDocument(system, "myXPCML.xpcml");
```

La sola differenza che intercorre tra la creazione di ProgramCallDocument XPCML e ProgramCallDocument PCML è che si passa al programma di creazione un documento XPCML invece di un documento PCML.

Nota: è necessario specificare .xpcml come estensione file per i file XPCML. L'utilizzo di .xpcml come estensione del file assicura che la classe ProgramCallDocument riconosca il file come XPCML. Se non si specifica un'estensione, ProgramCallDocument presuppone che il file sia PCML.

Utilizzo di XPCML per chiamare un programma sul server iSeries

Dopo aver creato un oggetto ProgramCallDocument, utilizzare qualsiasi metodo della classe ProgramCallDocument per gestire il documento XPCML. Ad esempio, chiamare un programma iSeries utilizzando ProgramCallDocument.callProgram() o modificare il valore di un parametro di immissione XPCML prima di chiamare il programma server utilizzando il metodo ProgramCallDocument.setValue appropriato.

Il seguente esempio mostra come creare un oggetto ProgramCallDocument per un file XPCML (denominato myXPCML.xpcml). Dopo la creazione dell'oggetto ProgramCallDocument, l'esempio chiama un programma (PROG1) specificato nel documento XPCML. In questo caso, la sola differenza tra l'utilizzo di XPCML e PCML è che l'esempio passa un file XPCML al programma di creazione ProgramCallDocument.

Dopo che l'applicazione legge e analizza un documento XPCML, il documento XPCML funziona esattamente come un documento PCML. A questo punto, XPCML può utilizzare qualsiasi metodo esistente utilizzato da PCML.

```
system = new AS400();

// Creare ProgramCallDocument in cui analizzare il file.
ProgramCallDocument xpcmlDoc = new ProgramCallDocument(system, "myXPCML.xpcml");

// Chiamare PROG1
boolean rc = xpcmlDoc.callProgram("PROG1");
```

Come ottenere i risultati della chiamata al programma come XPCML:

Dopo aver chiamato un programma server, è possibile utilizzare i metodi ProgramCallDocument.getValue per richiamare gli oggetti Java che rappresentano valori di parametro del programma. Inoltre, i seguenti metodi generateXPCML abilitano ProgramCallDocument a restituire i risultati di una chiamata al programma come XPCML:

- generateXPCML(String fileName): genera risultati in XPCML per l'intero file sorgente XPCML utilizzato per creare l'oggetto ProgramCallDocument. Memorizza l'XPCML in un file con il nome file specificato.
- generateXPCML(String pgmName, String fileName): genera risultati in XPCML solo per il programma specificato ed i relativi parametri. Memorizza XPCML in un file con il nome file specificato.
- generateXPCML(java.io.OutputStream outputStream): genera risultati in XPCML per l'intero file sorgente XPCML. Memorizza l'XPCML nell'oggetto OutputStream specificato.
- generateXPCML(String pgmName, java.io.OutputStream outputStream): genera risultati in XPCML solo per il programma specificato ed i relativi parametri. Memorizza l'XPCML nell'oggetto OutputStream specificato.

Per ulteriori informazioni sulla classe ProgramCallDocument, consultare la seguente pagina:

Informazioni javadoc ProgramCallDocument

Il seguente esempio mostra come è possibile creare un ProgramCallDocument XPCML, chiamare un programma iSeries e richiamare i risultati della chiamata al programma come XPCML.

"Esempio: richiamo dei risultati di una chiamata al programma come XPCML" a pagina 747

Inoltro dei valori di parametro come XPCML:

E' possibile impostare i valori per i parametri di programma nel file sorgente XPCML e passare i valori di parametro come XPCML. Quando l'oggetto ProgramCallDocument legge e analizza il documento XPCML, questo chiama automaticamente il metodo setValue appropriato per ogni parametro specificato in XPCML.

Utilizzare XPCML per passare valori di parametro esonera l'utente dal dover scrivere codice Java che imposta i valori di complicate strutture e schiere.

I seguenti esempi mostrano differenti modi di creare schiere e passare valori di parametro come XPCML:

"Esempio: inoltro dei valori di parametro come XPCML" a pagina 749

"Esempi: inoltro delle schiere di valori di parametro come XPCML" a pagina 751

Utilizzo di XPCML condensato:

Poiché XPCML è estensibile, è possibile definire nuovi tipi di parametro che estendono quelli specificati dallo schema XPCML. Condensando XPCML si estende lo schema XPCML in modo da creare nuove DTD (data type definition) che semplificano e migliorano la capacità di leggere e utilizzare i documenti XPCML.

La seguente discussione parte dal presupposto che si comprenda lo schema XPCML. Per ulteriori informazioni sullo schema XPCML, consultare la seguente pagina:

"Schema e sintassi XPCML" a pagina 421

Per condensare il sorgente XPCML esistente, si utilizza il metodo ProgramCallDocument.condenseXPCML, che genera quanto segue:
- Uno schema esteso che contiene nuove definizioni di tipo per ogni parametro nel sorgente XPCML esistente
- Nuovo sorgente XPCML che utilizza le definizioni di tipo fornite nello schema esteso

Per ulteriori informazioni su come condensare XPCML, consultare le seguenti pagine:

"Condensamento dei documenti XPCML esistenti"

"Esempio: utilizzo di XPCML condensato per creare un oggetto ProgramCallDocument" a pagina 755

"Esempio: come ottenere i risultati della chiamata al programma come XPCML condensato" a pagina 756

Condensamento dei documenti XPCML esistenti:

Il condensare documenti XPCML esistenti rende più facile la lettura e l'utilizzo del sorgente XPCML. Per creare XPCML condensato, utilizzare il metodo ProgramCallDocument.condenseXPCML. Per chiamare condenseXPCML(), fornire i seguenti parametri al metodo:

- Un flusso di immissione che rappresenta l'XPCML esistente
- Un flusso di emissione che rappresenta l'XPCML condensato
- Un flusso di emissione che rappresenta il nuovo, schema esteso
- Un nome per il nuovo schema nel formato appropriato (ad esempio, mySchema.xsd)

Per ulteriori informazioni su condenseXPCML() e sulla classe ProgramCallDocument, consultare la seguente pagina:

Informazioni javadoc ProgramCallDocument

ProgramCallDocument.condenseXPCML() è un metodo statico, il che significa che non è necessario creare un'istanza dell'oggetto ProgramCallDocument per chiamare il metodo.

Esempi

I seguenti esempi illustrano come condensare un documento XPCML esistente.

Il primo esempio è semplice ed include il sorgente XPCML originale, l'XPCML condensato risultante e lo schema esteso. Il secondo esempio è più lungo e maggiormente complesso, quindi include il codice Java che chiama condenseXPCML() e solo alcune delle definizioni di tipo appena generate nello schema esteso:

"Esempio: condensamento di un documento XPCML esistente" a pagina 752

"Esempio: condensamento di un documento XPCML esistente, incluso il codice Java" a pagina 753

Identificazione degli errori di analisi in XPCML:

Quando si convalidano documenti di schema XPCML, un programma di analisi XML a convalida completa può generare avvertenze, errori di analisi non irreversibili ed errori di analisi irreversibili.

Le avvertenze e gli errori di analisi non irreversibili non causano l'esito negativo dell'analisi. L'utente potrebbe voler esaminare le avvertenze e gli errori non irreversibili come aiuto per individuare problemi con il sorgente XPCML. Errori di analisi irreversibili fanno sì che l'analisi si concluda con un'eccezione.

Per visualizzare avvertenze ed errori del programma di analisi non irreversibili durante l'analisi di un documento XPCML, attivare la traccia nell'applicazione ed impostare la categoria di traccia su PCML.

Esempio

Un programma di analisi XML a convalida completa genera un errore per qualsiasi tipo di parametro numerico che non ha un valore. Il seguente esempio riporta il sorgente XPCML di esempio e il risultante errore di analisi non irreversibile:

Sorgente XPCML

```
<program name="prog1"/>
   <parameterList>
      <intParm name="parm1"/>
   </parameterList>
</program>
```

Errore risultante

```
Tue Mar 25 15:21:44 CST 2003  [Error]: cvc-complex-type.2.2: Element
'intParm' must have no element [children], and the value must be valid.
```

Per impedire la registrazione di questo tipo di errore, aggiungere l'attributo nil=true all'elemento intParm. L'attributo nil=true segnala al programma di analisi che l'elemento è stato lasciato intenzionalmente vuoto. Ecco il sorgente XPCML precedente con l'attributo nil=true aggiunto:

```
<program name="prog1"/>
   <parameterList>
      <intParm xsi:nil="true" name="parm1"/>
   </parameterList>
</program>
```

FAQ (Frequently asked questions)

Le FAQ (Frequently asked questions) di IBM Toolbox per Java forniscono risposte a domande relative all'ottimizzazione delle prestazioni di IBM Toolbox per Java, alla risoluzione di problemi, all'utilizzo di JDBC e altro:

- Le FAQ di IBM Toolbox per Java : rispondono a varie domande, incluso il miglioramento delle prestazioni, l'utilizzo di OS/400, la risoluzione dei problemi e altro.
- Le FAQ JDBC di IBM Toolbox per Java : rispondono a domande relative all'utilizzo di JDBC con Toolbox per Java

Suggerimenti per la programmazione

Questa sezione presenta una varietà di suggerimenti per l'utilizzo di IBM Toolbox per Java:

Chiusura del programma Java
Acquisire informazioni su come chiudere il programma Java in modo appropriato.

Utilizzo dei nomi del percorso IFS
Consultare le informazioni sull'utilizzo dei nomi di percorso IFS nei propri programmi. Questa sezione include nomi di percorso IFS, parametri e valori speciali.

Gestione dei collegamenti
Scoprire come utilizzare la classe AS400 per avviare e chiudere i collegamenti socket e come renderli compatibili con la specifica Enterprise JavaBean.

Utilizzo della JVM (Java virtual machine) di OS/400
Acquisire conoscenze sull'utilizzo delle classi di IBM Toolbox per Java sulla JVM di OS/400. Questa sezione illustra come accedere nel modo migliore alle risorse del server, come eseguire le classi e quali fattori di collegamento considerare.

Collegamento ad un ASP (auxiliary storage pool) indipendente
Acquisire informazioni sul collegamento ad un ASP indipendente. Un ASP indipendente è una raccolta di unità disco che è possibile mettere in linea o fuori linea indipendentemente dal memoria restante su un sistema.

Gestione degli errori durante l'utilizzo delle classi access di Toolbox per Java
Acquisire informazioni sull'utilizzo delle classi di eccezioni di IBM Toolbox per Java per gestire gli errori quando si utilizzano le classi access IBM Toolbox per Java nel programma.

Gestione degli errori durante l'utilizzo delle classi vaccess di IBM Toolbox per Java
Apprendere ad utilizzare le classi di eventi errore di IBM Toolbox per Java per gestire gli errori quando si utilizzano le classi vaccess IBM Toolbox per Java nel programma.

Utilizzo della classe Trace
Acquisire conoscenze su come utilizzare la classe Trace per registrare i punti traccia e i messaggi di diagnostica per facilitare la riproduzione e la diagnostica dei problemi nel proprio programma.

Ottimizzazione dei propri programmi
Acquisire informazioni su come ottimizzare i propri programmi per delle prestazioni migliori.

Come ottenere delle prestazioni migliori utilizzando JVM di OS/400
Consultare le informazioni sulle prestazioni potenziate, conseguenza dell'utilizzo di JVM di OS/400.

Gestione delle classi di IBM Toolbox per Java sul client
Acquisire conoscenze su come utilizzare la classe AS400ToolboxInstaller per gestire le classi IBM Toolbox per Java sul proprio client.

Miglioramento delle prestazioni del file JAR
Apprendere ad utilizzare la classe JarMaker di IBM Toolbox per Java per creare file JAR di IBM Toolbox per Java di minori dimensioni e più rapidamente caricabili.

Utilizzo del national language support di Java
Consultare le informazioni sull'utilizzo di IBM Toolbox per Java e dell'NLS di Java.

Come ottenere servizio e supporto
Utilizzare queste risorse per reperire servizi di supporto per IBM Toolbox per Java.

Chiusura del programma Java

Per assicurarsi che il programma venga chiuso in modo corretto, immettere System.exit(0) come ultima istruzione prima che il programma Java termini.

Nota: evitare di utilizzare System.exit(0) con i servlet poiché così facendo si chiude l'intera JVM (Java virtual machine).

IBM Toolbox per Java si collega al server grazie a sottoprocessi utente. A causa di ciò, un malfunzionamento nell'immissione di System.exit(0) potrebbe far si che il programma Java non venga chiuso correttamente.

L'utilizzo di System.exit(0) non è necessario, ma è una precauzione. Vi sono volte che è necessario utilizzare questo comando per uscire dal programma Java e l'utilizzo di System.exit(0) non comporta problemi quando non è necessario.

Nomi percorso IFS per gli oggetti server

Il programma Java deve utilizzare nomi IFS per riferirsi agli oggetti server, quali programmi, archivi, comandi o file di spool. Il nome IFS è il nome di un oggetto server nella forma in cui vi si accede nel file system della libreria dell'IFS sul server iSeries.

Il nome del percorso può essere costituito dai seguenti elementi:

Componente nome percorso	Descrizione
libreria	La libreria in cui risiede l'oggetto. La libreria è una parte **necessaria** di un nome percorso IFS.Il nome della libreria deve essere composto da un massimo di 10 caratteri seguito da **.lib**.

Componente nome percorso	Descrizione
oggetto	Il nome dell'oggetto rappresentato dal nome percorso IFS. L'oggetto è una parte **necessaria** di un nome percorso IFS.Il nome dell'oggetto deve essere composto da un massimo di 10 caratteri e seguito da **.type**, dove **type** è il tipo dell'oggetto.I tipi possono essere trovati richiedendo il parametro OBJTYPE sui comandi CL (Control Language) come il comando WRKOBJ (Gestione oggetti).
tipo	Il tipo dell'oggetto. Il tipo dell'oggetto deve essere specificato quando si specifica l'**oggetto**.(Si rimanda alla voce **oggetto**.)Il nome del tipo deve essere composto da un massimo di 6 caratteri.
membro	Il nome del membro rappresentato da questo nome percorso IFS. Il membro è una parte **facoltativa** di un nome percorso IFS.Può essere specificato solo quando il **tipo oggetto** è **FILE**.Il nome del membro deve contenere un massimo di 10 caratteri e deve essere seguito da **.mbr**.

Seguire queste istruzioni quando si determina e si specifica il nome IFS:

- La barra (/) è il carattere di separazione del percorso.
- L'indirizzario livello root, denominato QSYS.LIB, contiene la struttura della libreria del server.
- Gli oggetti che risiedono nella libreria server QSYS hanno il seguente formato:

```
/QSYS.LIB/object.type
```

- Gli oggetti che risiedono in altre librerie hanno il seguente formato:

```
/QSYS.LIB/library.LIB/object.type
```

- L'estensione tipo dell'oggetto è l'abbreviazione del server utilizzata per quel tipo di oggetto.

Per consultare un'elenco di questi tipi, immettere un comando CL che dispone di un tipo di oggetto come parametro e premere **F4** (Richiesta) per il tipo.Ad esempio, il comando WRKOBJ (Gestione oggetti) dispone di un parametro di tipo oggetto.

La tabella che segue è un elenco di alcuni tipi di oggetti comunemente utilizzati e l'abbreviazione per ogni tipo:

Tipo di oggetto	Abbreviazione
comando	.CMD
coda dati	.DTAQ
file	.FILE
risorsa font	.FNTRSC
definizione modulo	.FORMDF
libreria	.LIB
membro	.MBR
sovrapposizione	.OVL
definizione di pagina	.PAGDFN
segmento di pagina	.PAGSET
programma	.PGM
coda di emissione	.OUTQ
file di spool	.SPLF

Utilizzare le descrizioni che seguono per determinare come specificare i nomi percorso IFS:

Nome IFS	Descrizione
/QSYS.LIB/MY_LIB.LIB/MY_PROG.PGM	Programma MY_PROG nella libreria MY_LIB sul server
/QSYS.LIB/MY_LIB.LIB/MY_QUEUE.DTAQ	Coda dati MY_QUEUE nella libreria MY_LIB sul server
/QSYS.LIB/YEAR1998.LIB/MONTH.FILE/JULY.MBR	Membro JULY nel file MONTH nella libreria YEAR1998 sul server

Valori speciali IFS

Varie classi IBM Toolbox per Java riconoscono valori speciali nei nomi percorso IFS. Il formato tradizionale per questi valori speciali (utilizzati in una riga comandi iSeries) inizia con un asterisco (***ALL**). Tuttavia, in un programma Java che utilizza le classi IBM Toolbox per Java, il formato per tali valori speciali inizia e termina con i simboli di percentuale (**%ALL%**).

Nota: nell'IFS, un asterisco corrisponde a un carattere jolly.

La tabella che segue mostra quali tra questi valori speciali vengono riconosciuti dalle classi IBM Toolbox per Java per componenti nome percorso particolari. La tabella mostra inoltre come il formato tradizionale per questi valori speciali differisce dal formato utilizzato nelle classi IBM Toolbox per Java.

Componente nome percorso	Formato tradizionale	Formato IBM Toolbox per Java
Nome archivio	*ALL	%ALL%
	*ALLUSR	%ALLUSR%
	*CURLIB	%CURLIB%
	*LIBL	%LIBL%
	*USRLIBL	%USRLIBL%
Nome oggetto	*ALL	%ALL%
Nome membro	*ALL	%ALL%
	*FILE	%FILE%
	*FIRST	%FIRST%
	*LAST	%LAST%

Consultare la classe QSYSObjectPathName per informazioni relative alla creazione e all'analisi di nomi IFS.

Per ulteriori informazioni relative ai concetti IFS, consultare Concetti IFS.

Gestione dei collegamenti

E' importante poter creare, avviare e chiudere i collegamenti al server. Le informazioni seguenti contengono i concetti fondamentali per gestire i collegamenti al server ed offrono anche alcuni esempi di codice.

Per collegarsi a un server iSeries, il programma Java deve creare un oggetto AS400. L'oggetto AS400 contiene al massimo un collegamento al socket per ogni tipo di server iSeries. Un servizio corrisponde ad un lavoro sul server ed è l'interfaccia per i dati sul server.

Nota: quando si crea un EJB (Enterprise JavaBeans), attenersi alla specifica EJB che non consente sottoprocessi durante il collegamento. Anche se la disattivazione del supporto del sottoprocesso di IBM Toolbox per Java può rallentare l'applicazione, è richiesta per attenersi alla specifica EJB.

Ogni collegamento a ciascun server ha un proprio lavoro su iSeries. Un server differente supporta ognuna delle seguenti operazioni:

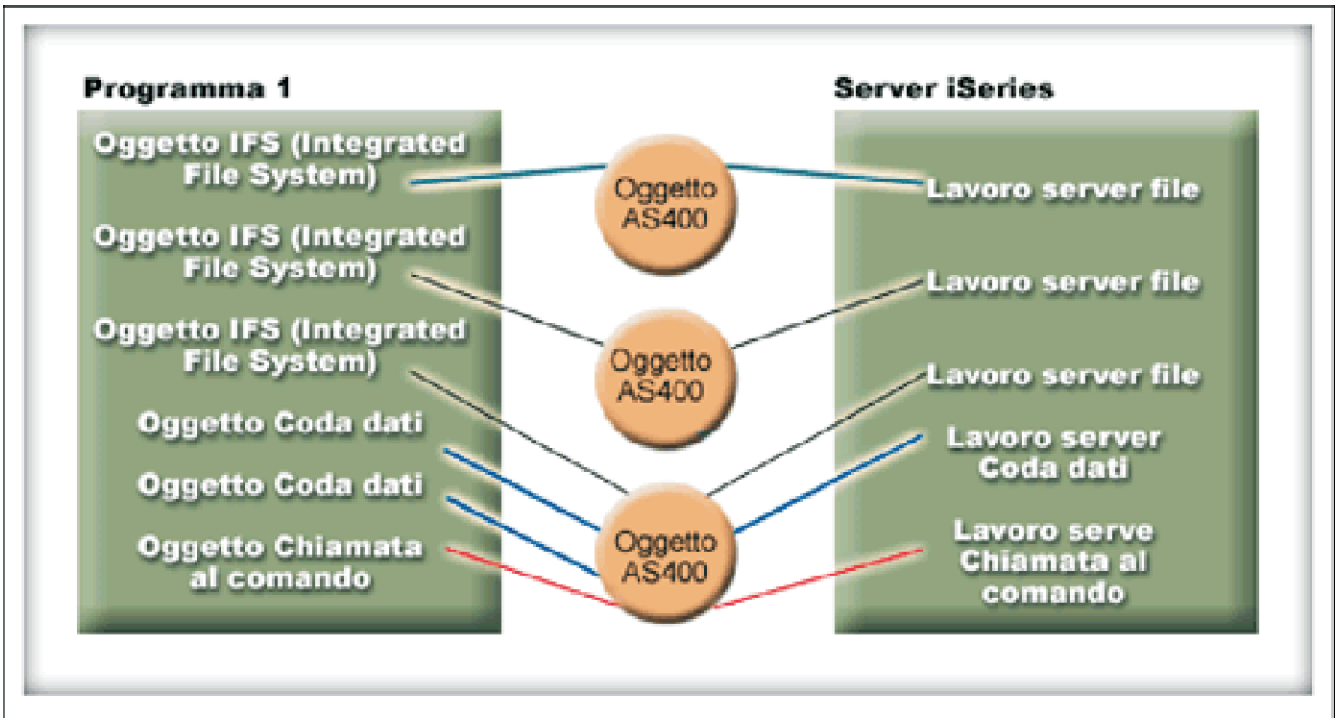
- JDBC
- Chiamata al programma e chiamata al comando
- IFS (Integrated file system)
- Stampa
- Coda dati
- Accesso al livello record

Nota:

- Le classi print utilizzano un collegamento socket per l'oggetto di AS400 se l'applicazione non effettua due operazioni che richiedono contemporaneamente il server di stampa della rete.
- Una classe print crea collegamenti socket aggiuntivi al server di stampa della rete se necessario.Le conversazioni supplementari vengono scollegate se non sono utilizzate per 5 minuti.

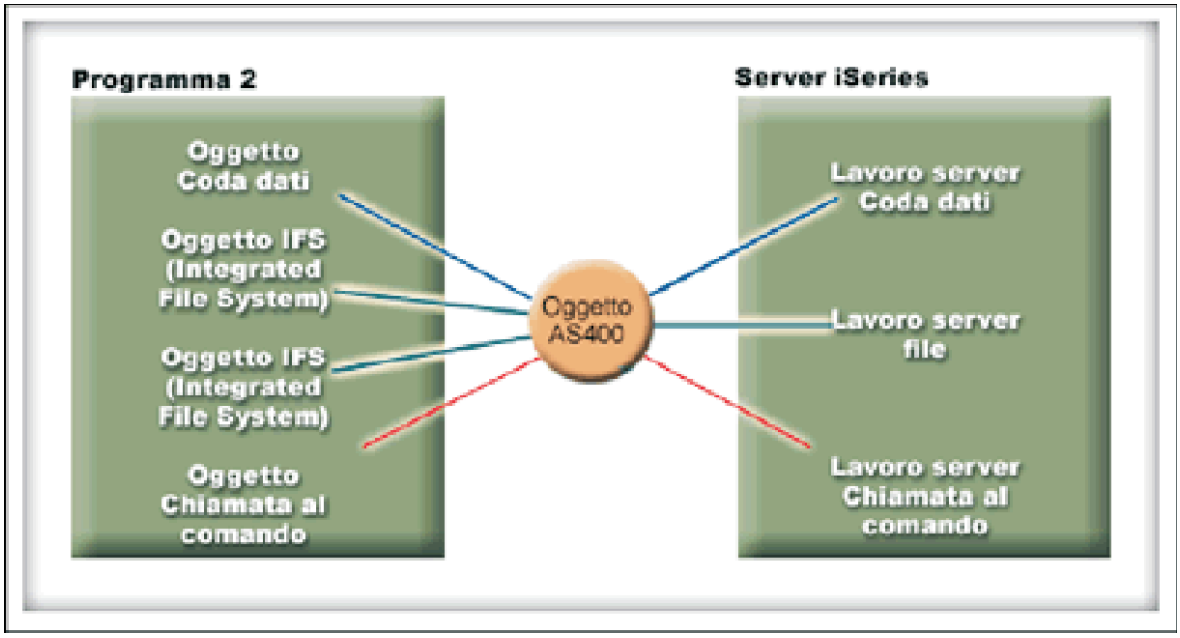
Il programma Java può controllare il numero di collegamenti a iSeries. Per ottimizzare le prestazioni delle comunicazioni, un programma Java può creare più oggetti AS400 per lo stesso server come mostrato nella Figura 1. In questo modo vengono creati più collegamenti socket al server iSeries.

Figura 1: esempio di programma Java che crea più oggetti AS400 e collegamenti socket per lo stesso server iSeries



Per conservare le risorse del server, creare solo un oggetto AS400 come mostrato in Figura 2. Questa operazione consente di ridurre il numero di collegamenti, il che riduce a sua volta la quantità di risorse utilizzate sul server.

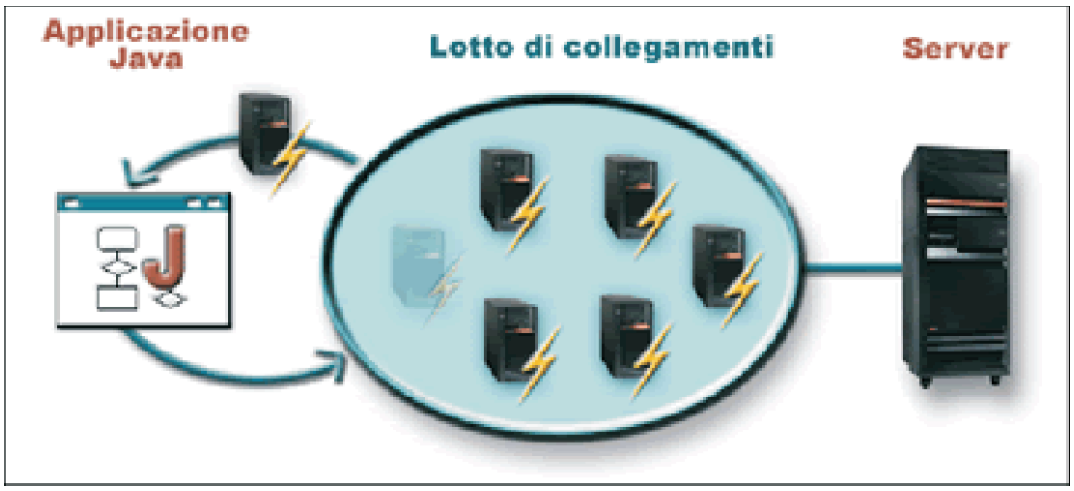
Figura 2: esempio di programma Java che crea un singolo oggetto AS400 e un collegamento socket per lo stesso server iSeries



Nota: anche se la creazione di più collegamenti aumenta la quantità di risorse utilizzate nel sistema, la creazione di più collegamenti può risultare vantaggiosa. Avere più collegamenti consente al programma Java di elaborare operazioni in parallelo, il che può portare ad un migliore rendimento (transazioni-per-secondo) e velocizzare l'applicazione.

E' anche possibile scegliere di utilizzare un lotto di collegamenti per gestire i collegamenti, come mostrato nella Figura 3. Questo approccio riduce la quantità di tempo necessaria per collegarsi ad iSeries riutilizzando un collegamento precedentemente stabilito per l'utente.

Figura 3: esempio di programma Java che richiama un collegamento da AS400ConnectionPool a un server iSeries



I seguenti esempi mostrano come creare e utilizzare gli oggetti AS400:

Esempio 1: Nel seguente esempio, vengono creati due oggetti CommandCall che inviano i comandi allo stesso server. Poiché gli oggetti CommandCall utilizzano lo stesso oggetto AS400, viene creato solo il collegamento al server.

```
            // Creare un oggetto AS400.
     AS400 sys = new AS400("mySystem.myCompany.com");

            // Creare due oggetti di chiamata al comando che utilizzano
            // lo stesso oggetto AS400.
     CommandCall cmd1 = new CommandCall(sys,"myCommand1");
     CommandCall cmd2 = new CommandCall(sys,"myCommand2");

            // Eseguire i comandi.
Viene stabilito un collegamento alla
            // prima esecuzione del comando. Poiché utilizzano lo stesso
            // oggetto AS400, il secondo oggetto di comando utilizzerà il
            // collegamento stabilito dal primo comando.
     cmd1.run();
     cmd2.run();
```

Esempio 2: nel seguente esempio, vengono creati due oggetti CommandCall che inviano i comandi allo stesso server iSeries. Poiché gli oggetti CommandCall utilizzano differenti oggetti AS400, vengono creati due collegamenti al server.

```
            // Creare due oggetti AS400 nello stesso server.
     AS400 sys1 = new AS400("mySystem.myCompany.com");
     AS400 sys2 = new AS400("mySystem.myCompany.com");

            // Creare due oggetti chiamata al comando.
Utilizzano
            // diversi oggetti AS400.
     CommandCall cmd1 = new CommandCall(sys1,"myCommand1");
     CommandCall cmd2 = new CommandCall(sys2,"myCommand2");

            // Eseguire i comandi.
Viene stabilito un collegamento alla
            // prima esecuzione del comando. Poiché il secondo
            // comando utilizza un oggetto AS400 diverso, viene stabilito un secondo
            // collegamento quando viene eseguito il secondo comando.
     cmd1.run();
     cmd2.run();
```

Esempio 3: nel seguente esempio, vengono creati un oggetto CommandCall e un oggetto IFSFileInputStream utilizzando lo stesso oggetto AS400. Poiché gli oggetti CommandCall e IFSFileInput Stream utilizzano differenti servizi sul server iSeries, vengono creati due collegamenti.

```
            // Creare un oggetto AS400.
     AS400 newConn1 = new AS400("mySystem.myCompany.com");

            // Creare un oggetto chiamata al comando.
     CommandCall cmd = new CommandCall(newConn1,"myCommand1");

            // Creare l'oggetto file.
Tale creazione causerà il collegamento
            // dell'oggetto AS400 al servizio file.
     IFSFileInputStream file = new IFSFileInputStream(newConn1,"/myfile");

            // Eseguire il comando.
Viene stabilito un collegamento al
            // servizio comando quando viene eseguito il comando.
     cmd.run();
```

Esempio 4: nel seguente esempio, viene utilizzato un AS400ConnectionPool per richiamare un collegamento iSeries. Questo esempio (come l'Esempio 3 precedente) non specifica un servizio, quindi viene effettuato il collegamento al servizio del comando quando viene eseguito il comando.

```
            // Creare un AS400ConnectionPool.
     AS400ConnectionPool testPool1 = new AS400ConnectionPool();
            // Creare un collegamento.
     AS400 newConn1 = testPool1.getConnection("myAS400", "myUserID", "myPassword");
            // Creare un oggetto chiamata al comando che usa l'oggetto AS400.
     CommandCall cmd = new CommandCall(newConn1,"myCommand1");
            // Eseguire il comando.
Viene stabilito un collegamento al
            // servizio comando quando viene eseguito il comando.
     cmd.run();
            // Restituire il collegamento al lotto.
     testPool1.returnConnectionToPool(newConn1);
```

Esempio 5: il seguente esempio utilizza AS400ConnectionPool per effettuare il collegamento ad un particolare servizio quando viene richiesto il collegamento dal lotto. In questo modo si elimina il tempo richiesto per il collegamento al servizio quando il comando viene eseguito (vedere Esempio 4 precedente). Se il collegamento viene restituito al lotto, la chiamata successiva per ottenere un collegamento può restituire lo stesso oggetto di collegamento. Ciò significa che non è necessario ulteriore tempo di collegamento, né per la creazione né per l'utilizzo.

```
            // Creare un AS400ConnectionPool.
     AS400ConnectionPool testPool1 = new AS400ConnectionPool();
            // Creare un collegamento al servizio AS400.COMMAND. (Utilizzare le costanti del numero servizio
            // definite nella classe AS400 (FILE, PRINT, COMMAND, DATAQUEUE e così via.))
     AS400 newConn1 = testPool1.getConnection("myAS400", "myUserID", "myPassword", AS400.COMMAND);
            // Creare un oggetto chiamata al comando che usa l'oggetto AS400.
     CommandCall cmd = new CommandCall(newConn1,"myCommand1");
            // Eseguire il comando.
E' stato già stabilito un collegamento
            // al servizio comando.
     cmd.run();
            // Restituire il collegamento al lotto.
     testPool1.returnConnectionToPool(newConn1);
            // Richiamare un altro collegamento al servizio comando. In questo caso, verrà restituito lo stesso
            // collegamento come in precedenza, indicando che non sarà necessario altro tempo di collegamento ora
            // o quando viene utilizzato il servizio comando.
     AS400 newConn2 = testPool1.getConnection("myAS400", "myUserID", "myPassword", AS400.COMMAND);
```

Avvio e chiusura dei collegamenti

Il programma Java può controllare l'avvio e la chiusura di un collegamento. Per impostazione predefinita, un collegamento viene avviato quando sono necessarie informazioni dal server. E' possibile controllare esattamente quando viene effettuato il collegamento richiamando il metodo connectService() sull'oggetto AS400 per precollegarsi al server.

Utilizzando un AS400ConnectionPool, è possibile creare un collegamento precollegato ad un servizio senza richiamare il metodo connectService(), come nell'Esempio 5 precedente.

I seguenti esempi mostrano i programmi Java che si collegano e si scollegano da iSeries.

Esempio 1: questo esempio mostra come precollegarsi ad iSeries:

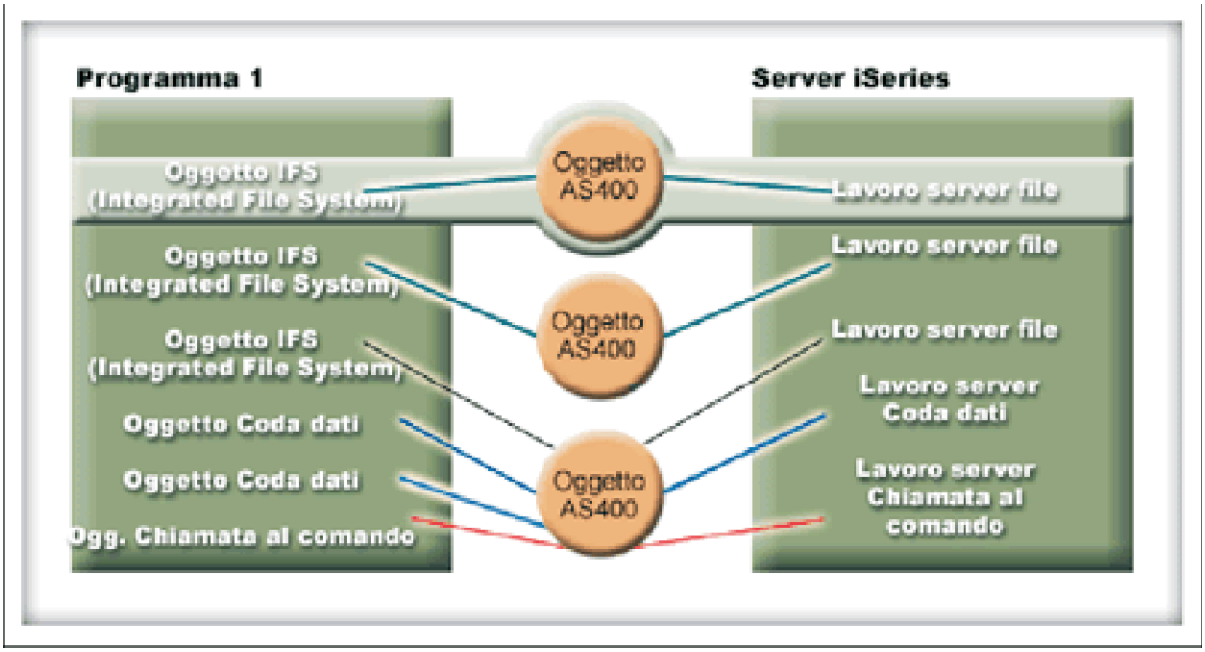
```
            // Creare un oggetto AS400.
     AS400 system1 = new AS400("mySystem.myCompany.com");

            // Stabilire un collegamento al servizio comando. Stabilire tale collegamento ora e non
            // quando i dati vengono inviati per la prima volta
            // al servizio comando. Questa operazione è facoltativa poiché
            // l'oggetto AS400 si collegherà quando necessario.
     system1.connectService(AS400.COMMAND);
```

Esempio 2: una volta avviato il collegamento, il programma Java è responsabile dello scollegamento, che viene eseguito implicitamente dall'oggetto AS400 o esplicitamente tramite il programma Java. Un

programma Java si scollega richiamando il metodo disconnectService() sull'oggetto AS400. Per migliorare le prestazioni, il programma Java dovrebbe scollegarsi solo quando il programma viene terminato con un servizio. Se il programma Java si scollega prima di essere terminato con un servizio, l'oggetto AS400 si ricollega (se possibile) quando vengono richiesti i dati relativi al servizio.

La Figura 4 mostra come, interrompendo il primo collegamento dell'oggetto IFS si chiude solo quella singola istanza del collegamento dell'oggetto AS400, non tutti i collegamenti dell'oggetto IFS.

Figura 4: il singolo oggetto che utilizza il proprio servizio per un'istanza di un oggetto AS400 viene scollegato

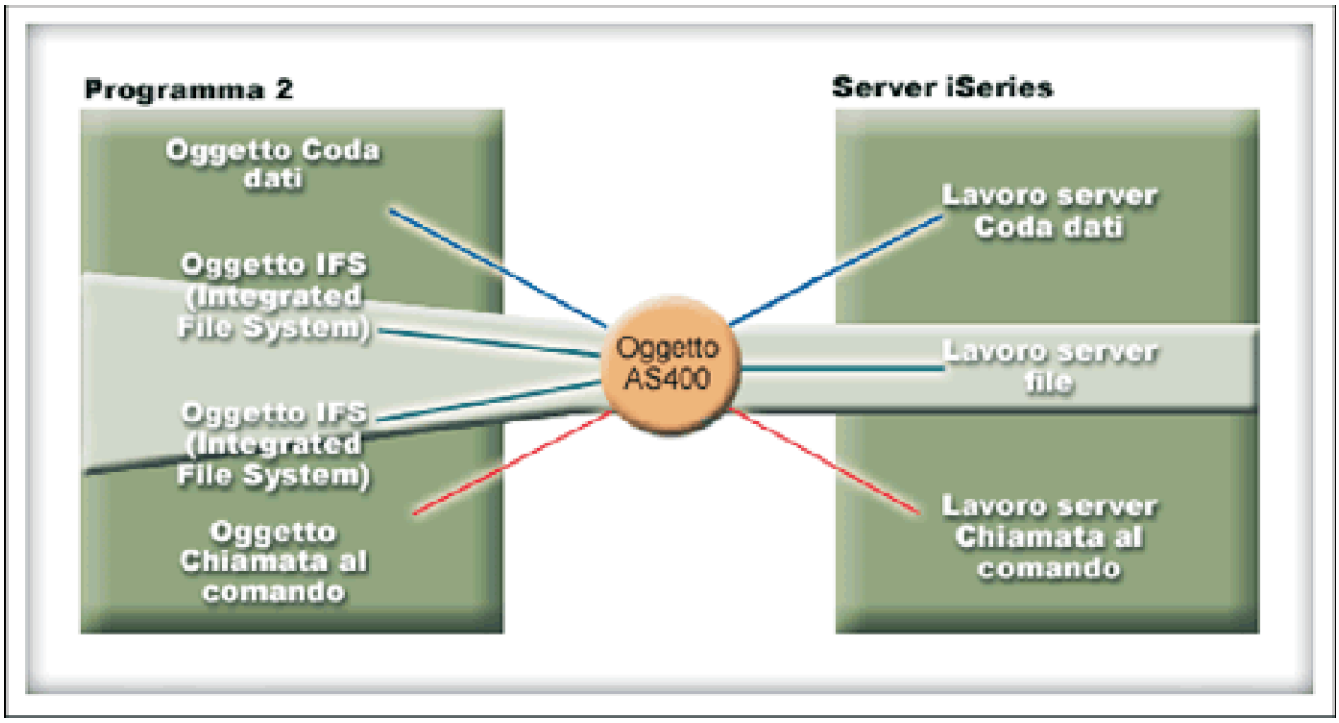


Questo esempio mostra come un programma Java termina un collegamento:

```
        // Creare un oggetto AS400.
AS400 system1 = new AS400("mySystem.myCompany.com");

        // ... utilizzare la chiamata al comando per inviare diversi comandi
        // al server. Poiché connectService() non è stato richiamato,
        // l'oggetto AS400 si collegherà automaticamente
        // quando viene eseguito il primo comando.

        // Eseguiti tutti i comandi di invio quindi disattivare
        // il collegamento.
system1.disconnectService(AS400.COMMAND);
```

Esempio 3: più oggetti che utilizzano lo stesso servizio e condividono lo stesso oggetto AS400 condividono un collegamento. Lo scollegamento chiude il collegamento per tutti gli oggetti che utilizzano lo stesso servizio per ogni istanza di un oggetto AS400, come mostrato nella Figura 5.

Figura 5: tutti gli oggetti che utilizzano lo stesso servizio per un'istanza di un oggetto AS400 vengono scollegati



Ad esempio, due oggetti CommandCall utilizzano lo stesso oggetto AS400. Quando viene richiamato disconnectService(), il collegamento viene terminato per entrambi gli oggetti CommandCall. Quando viene richiamato il metodo run() per il secondo oggetto CommandCall, l'oggetto AS400 deve ricollegarsi al servizio:

```
            // Creare un oggetto AS400.
     AS400 sys = new AS400("mySystem.myCompany.com");

            // Creare due oggetti chiamata al comando.
     CommandCall cmd1 = new CommandCall(sys,"myCommand1");
     CommandCall cmd2 = new CommandCall(sys,"myCommand2");

            // Eseguire il primo comando
     cmd1.run();

            // Scollegarsi dal servizio comando.
     sys.disconnectService(AS400.COMMAND);

            // Eseguire il secondo comando. L'oggetto AS400
            // deve ricollegarsi al server.
     cmd2.run();

            // Scollegarsi dal servizio comando.
Questo
            // è il posto giusto per scollegarsi.
     sys.disconnectService(AS400.COMMAND);
```

Esempio 4: non tutte le classi IBM Toolbox per Java si ricollegano automaticamente. Alcune chiamate al metodo contenute nelle classi IFS non si ricollegano poiché il file potrebbe essere stato modificato. Mentre il file era scollegato, qualche altro processo può avere cancellato il file o modificato il contenuto. Nell'esempio seguente, due oggetti file utilizzano lo stesso oggetto AS400. Quando viene richiamato disconnectService(), il collegamento viene terminato per entrambi gli oggetti file. Il metodo read() per il secondo oggetto IFSFileInputStream non riesce poiché non esiste più il collegamento al server.

```
            // Creare un oggetto AS400.
     AS400 sys = new AS400("mySystem.myCompany.com");

            // Creare due oggetti file. Viene stabilito un collegamento
```

```
            // al server quando viene creato il primo
            // oggetto. Il secondo oggetto utilizza il collegamento
            // creato dal primo oggetto.
     IFSFileInputStream file1 = new IFSFileInputStream(sys,"/file1");
     IFSFileInputStream file2 = new IFSFileInputStream(sys,"/file2");

            // Leggere dal primo file, poi chiuderlo.
          int i1 = file1.read();
          file1.close();

            // Scollegarsi dal servizio file.
     sys.disconnectService(AS400.FILE);

            // Tentare di leggere il secondo file. Tale operazione non
            // riesce perché il collegamento al servizio file non
            // esiste più. Il programma deve scollegarsi
            // successivamente o fare in modo che il secondo file
            // utilizzi un oggetto AS400 diversi (cosa che fa in modo che
            // abbia un proprio collegamento).
     int i2 = file2.read();

            // Chiudere il secondo file.
     file2.close();

            // Scollegarsi dal servizio file.
Questo
            // è il posto giusto per scollegarsi.
     sys.disconnectService(AS400.FILE);
```

Descrizione lunga della Figura 1: programma Java che crea più oggetti AS400 e collegamenti socket per lo stesso server iSeries (rzahh549.gif)

In IBM Toolbox per Java: gestire i collegamenti

Questa figura illustra come un programma Java può creare più oggetti AS400, ognuno dei quali ha un collegamento socket separato allo stesso server iSeries. La creazione di più collegamenti aumenta la quantità di risorsa utilizzata nel sistema.

Descrizione

La figura è composta da quanto segue:

- Un rettangolo sulla sinistra rappresenta un programma Java
- Un rettangolo sulla destra rappresenta un server iSeries
- Tre piccoli cerchi allineati verticalmente tra i rettangoli rappresentano gli oggetti AS400 creati dal programma Java che si collegano al server iSeries

Il programma Java (il rettangolo di sinistra) contiene diversi tipi di oggetti che utilizzano gli oggetti AS400 (i tre piccoli cerchi). Le linee da questi oggetti di programma si collegano agli oggetti AS400. Ogni oggetto di programma richiede l'utilizzo di un solo oggetto AS400. Tuttavia, diversi oggetti di programma di vari tipi possono utilizzare un singolo oggetto AS400 per collegarsi all'iSeries.

Le linee dagli oggetti AS400 si collegano ai servizi contenuti nel server iSeries (il rettangolo di destra). I servizi rispecchiano appropriatamente gli oggetti di programma. L'elenco seguente descrive come più oggetti programma, tre oggetti AS400 ed i servizi del server iSeries si collegano reciprocamente:

- Un oggetto di programma IFS si collega ad un primo oggetto AS400, il quale si collega al lavoro server del file
- Un altro oggetto di programma IFS si collega al secondo oggetto AS400, il quale si collega ad un secondo lavoro server del file

- Al terzo oggetto AS400 sono collegati più oggetti di programma: un terzo oggetto IFS, due oggetti coda dati e un oggetto chiamata al comando. L'oggetto AS400 quindi si collega ai seguenti servizi sul server iSeries: un terzo lavoro server file (per l'oggetto IFS), un lavoro server coda dati (per gli oggetti coda dati) ed un lavoro server chiamata comando (per l'oggetto chiamata comando).

Descrizione lunga della Figura 2: programma Java che crea un singolo oggetto AS400 ed un collegamento socket per lo stesso server iSeries (rzahh552.gif)

In IBM Toolbox per Java: gestire i collegamenti

Questa figura mostra come avendo più oggetti di programma che condividono un singolo oggetto AS400 si riduce il numero di collegamenti e la quantità di risorsa utilizzata sul sistema.

Descrizione

La figura è composta da quanto segue:

- Un rettangolo sulla sinistra rappresenta un programma Java
- Un rettangolo sulla destra rappresenta un server iSeries
- Un piccolo cerchio posto tra i rettangoli rappresenta un singolo oggetto AS400 creato dal programma Java che si collega al server iSeries

Il programma Java (il rettangolo di sinistra) contiene diversi tipi di oggetti che utilizzano l'oggetto AS400. Le linee da questi oggetti di programma si collegano all'oggetto AS400 (il cerchio piccolo).

Linee che partono dall'oggetto AS400 si collegano ai servizi contenuti nel server iSeries (il rettangolo di destra). Ogni servizio rappresenta un collegamento separato sul server. I servizi rispecchiano appropriatamente gli oggetti di programma:

- Un oggetto di programma coda dati utilizza l'oggetto AS400, il quale si collega ad un lavoro server coda dati.
- Due oggetti IFS (integrated file system) utilizzano l'oggetto AS400, il quale si collega ad un lavoro server del file singolo.
- Un oggetto chiamata al programma utilizza l'oggetto AS400, il quale si collega al lavoro server chiamata al comando

Tutti questi oggetti di programma utilizzano lo stesso oggetto AS400, il quale crea collegamenti separati per ogni lavoro che viene eseguito sul server. In questo caso, dal momento che gli oggetti IFS utilizzano lo stesso oggetto AS400, essi condividono lo stesso collegamento ad un lavoro sul server.

Descrizione lunga della Figura 3: programma Java che richiama un collegamento da un AS400ConnectionPool ad un server iSeries (rzahh506.gif)

In IBM Toolbox per Java: gestire i collegamenti

Questa figura illustra come utilizzare un lotto di collegamenti per gestire i collegamenti ad un server iSeries.

Descrizione

La figura è composta da quanto segue:

- Un'immagine rettangolare sulla sinistra rappresenta un'applicazione Java.
- Una raffigurazione di un server iSeries sulla destra che rappresenta il server a cui l'applicazione Java desidera collegarsi.
- Un ovale tra le immagini sulla sinistra e sulla destra rappresenta un oggetto AS400ConnectionPool.

All'interno di AS400ConnectionPool (l'ovale) vi sono più collegamenti. Ogni collegamento è un oggetto AS400, rappresentato come una piccola immagine di un server iSeries con un freccia a forma di fulmine su di esso:

- Una freccia circolare collega il programma Java (il rettangolo sulla sinistra) al lotto di collegamenti (l'ovale).
- Una linea singola collega il server iSeries (l'immagine sulla destra) al lotto di collegamenti (l'ovale).

La metà superiore della freccia circolare punta dall'AS400ConnectionPool (l'ovale) all'applicazione Java (l'immagine rettangolare sulla sinistra). Sulla freccia si trova uno dei collegamenti dal lotto e una delle immagini di collegamento nel lotto di collegamenti è scomparsa. Ciò rappresenta un collegamento dal lotto di collegamenti richiesto e utilizzato dall'applicazione Java.

La metà inferiore della freccia circolare punta dall'applicazione Java al lotto di collegamenti. Ciò rappresenta la restituzione da parte dell'applicazione Java del collegamento al lotto di collegamenti.

La linea singola che collega il server iSeries (l'immagine sulla destra) all'AS400ConnectionPool (l'ovale) rappresenta i collegamenti socket al server.

Descrizione lunga della Figura 4: come un oggetto singolo che utilizza il proprio servizio per un'istanza di un oggetto AS400 viene scollegato (rzahh550.gif)

In IBM Toolbox per Java: gestire i collegamenti

Questa figura illustra come la chiusura di un collegamento per un oggetto AS400 singolo non ha effetti sui collegamenti di altri oggetti AS400.

Descrizione

La figura è composta dagli stessi elementi della Figura 1:

- Un rettangolo sulla sinistra rappresenta un programma Java
- Un rettangolo sulla destra rappresenta un server iSeries
- Tre piccoli cerchi allineati verticalmente tra i rettangoli rappresentano gli oggetti AS400 creati dal programma Java che si collegano al server iSeries

Il programma Java (il rettangolo di sinistra) contiene diversi tipi di oggetti che utilizzano gli oggetti AS400 (la linea verticale dei tre piccoli cerchi). Le linee da questi oggetti di programma si collegano agli oggetti AS400. Ogni oggetto di programma richiede l'utilizzo di un solo oggetto AS400. Tuttavia, diversi oggetti di programma di vari tipi possono utilizzare un singolo oggetto AS400 per collegarsi all'iSeries.

Le linee dagli oggetti AS400 si collegano ai servizi contenuti nel server iSeries (il rettangolo di destra). I servizi rispecchiano appropriatamente gli oggetti di programma. Per una descrizione specifica dei differenti oggetti programma, oggetti AS400 e servizi server iSeries, consultare la descrizione relativa alla Figura 1.

Il collegamento per un oggetto AS400 singolo è evidenziato per mostrare come la sua chiusura non abbia effetti sui collegamenti degli altri oggetti AS400. Tale collegamento è composto da un oggetto IFS, dall'oggetto AS400 che utilizza, dal servizio server iSeries associato (un lavoro server file) e dalle linee che collegano tutti questi elementi. Nessuno degli altri oggetti di programma, linee, oggetti AS400 o servizi sono evidenziati.

Descrizione lunga della Figura 5: come tutti gli oggetti che utilizzano lo stesso servizio per un'istanza di un oggetto AS400 vengono scollegati (rzahh551.gif)

In IBM Toolbox per Java: gestire i collegamenti

Questa figura illustra come lo scollegamento di un servizio per un oggetto AS400 singolo provochi lo scollegamento di tutti gli oggetti che condividono tale servizio per quell'istanza dell'oggetto AS400.

Descrizione

La figura è composta dagli stessi elementi della Figura 2:

- Un rettangolo sulla sinistra rappresenta un programma Java
- Un rettangolo sulla destra rappresenta un server iSeries
- Un piccolo cerchio posto tra i rettangoli rappresenta un singolo oggetto AS400 creato dal programma Java che si collega al server iSeries

Il programma Java (il rettangolo di sinistra) contiene diversi tipi di oggetti che utilizzano l'oggetto AS400 (il cerchio piccolo). Le linee da questi oggetti di programma si collegano all'oggetto AS400.

Linee che partono dall'oggetto AS400 si collegano ai servizi contenuti nel server iSeries (il rettangolo di destra). I servizi rispecchiano appropriatamente gli oggetti di programma. Per una descrizione specifica dei differenti oggetti di programma, dell'oggetto AS400 e dei servizi server iSeries, consultare la descrizione relativa alla Figura 2 .

Due oggetti programma IFS utilizzano gli oggetti AS400, che si collegano ad un lavoro server file sul server iSeries. Questi due oggetti di programma, il servizio associato e le linee che li collegano sono evidenziati. Lo scollegamento del servizio ha effetto solo sugli elementi evidenziati, scollegando in effetti entrambi gli oggetti di programma. Nessuno degli altri oggetti di programma, servizi, collegamenti o l'oggetto AS400 viene interessato.

JVM (Java Virtual Machine) OS/400

Le classi IBM Toolbox per Java si eseguono su JVM (Java virtual machine) di IBM Developer Kit for Java (OS/400). In effetti, le classi vengono eseguite su ogni piattaforma che supporti le specifiche Java 2 Software Development Kit (J2SDK).

Quando si eseguono le classi IBM Toolbox per Java su JVM OS/400, effettuare quanto segue:

- Scegliere se utilizzare la JVM OS/400 o le classi IBM Toolbox per Java per accedere alle risorse del server iSeries in esecuzione nella JVM OS/400.
- Consultare Esecuzione delle classi IBM Toolbox per Java sulla JVM OS/400.
- Acquisire informazioni circa l'impostazione del nome di sistema, dell'ID utente e della parola d'ordine sulla JVM OS/400.

Per ulteriori informazioni sul supporto server iSeries per differenti piattaforme Java, consultare Support for multiple JDKs.

Confronto tra JVM OS/400 e le classi di IBM Toolbox per Java

L'utente ha sempre a disposizione almeno due modi per accedere alle risorse del server iSeries quando il programma Java è in esecuzione sulla JVM (Java virtual machine) IBM Developer Kit per Java (OS/400). E' possibile utilizzare uno o l'altra delle seguenti interfacce:

- Funzioni incorporate in Java
- Una classe IBM Toolbox per Java

Al momento di decidere quale interfaccia utilizzare, è necessario considerare i seguenti fattori:

- **Ubicazione** - L'ubicazione nella quale si esegue un programma è il fattore più importante nel decidere quale serie di interfacce utilizzare.Il programma:
  - Viene eseguito solo sul client?

– Viene eseguito solo sul server?
– Viene eseguito sia sul client che sul server, ma in entrambi i casi la risorsa è una risorsa del server iSeries?
– Viene eseguito sulla JVM OS/400 ed accede a risorse su un altro server iSeries?
– Viene eseguito su diversi tipi di server?

Se il programma viene eseguito sia su client che sul server (incluso un server iSeries come client di un secondo server iSeries) ed accede solo alle risorse del server iSeries, potrebbe essere più indicato l'utilizzo delle interfacce IBM Toolbox per Java.

Se il programma deve accedere a dati su molti tipi di server, potrebbe essere più indicato l'utilizzo delle JNI (Java Native Interface).

- **Coerenza / Portabilità** - La capacità di eseguire classi IBM Toolbox per Java sui server iSeries significa che le stesse interfacce possono essere utilizzate sia per i programmi client che per i programmi server.Quando si deve apprendere il funzionamento di una sola interfaccia sia per i programmi del client che per i programmi del server, è possibile essere più produttivi.

  Tuttavia, la scrittura sulle interfacce di IBM Toolbox per Java rende il programma più difficilmente compatibile con i **server**.

  Se il programma deve operare deve operare su un server iSeries ed anche su altri server, potrebbe essere più indicato l'utilizzo delle funzioni incorporate in Java.
- **Complessità** - L'interfaccia IBM Toolbox per Java viene creata soprattutto per facilitare l'accesso ad una risorsa del server iSeries.Spesso, l'unica alternativa all'utilizzo dell'interfaccia di IBM Toolbox per Java è scrivere un programma che abbia accesso alla risorsa e che comunichi con tale programma tramite JNI (Java Native Interface).

  E' necessario decidere se è più importante avere una maggiore neutralità di Java e scrivere un programma per accedere alla risorsa, oppure utilizzare l'interfaccia di IBM Toolbox per Java, che è meno compatibile.
- **Funzione** - L'interfaccia IBM Toolbox per Java spesso fornisce più funzioni rispetto all'interfaccia Java.Ad esempio, la classe IFSFileOutputStream del programma su licenza IBM Toolbox per Java possiede più funzioni rispetto alla classe FileOutputStream di java.io. Tuttavia, l'utilizzo di IFSFileOutputStream rende il programma specifico per i server iSeries. Si perde la compatibilità **server** utilizzando la classe IBM Toolbox per Java.

  L'utente deve decidere se la portabilità è più importante o se vuole il vantaggio di una ulteriore funzione.
- **Risorsa** - Quando sono in esecuzione sulla JVM OS/400, molte delle classi IBM Toolbox per Java effettuano ancora richieste tramite i server host. Pertanto, un secondo lavoro (il lavoro server) invia la richiesta di accesso ad una risorsa.

  Questa richiesta può impiegare più risorse di una JNI (Java Native Interface) che viene eseguita all'interno del lavoro del programma Java.
- **Server iSeries come client** - Se il programma viene eseguito su un server iSeries ed accede ai dati su un secondo server iSeries, la scelta migliore può essere quella di utilizzare le classi IBM Toolbox per Java.Queste classi forniscono un facile accesso alla risorsa di un secondo server iSeries.

  Un esempio di ciò è l'accesso alla coda dati. Le interfacce coda dati del programma su licenza di IBM Toolbox per Java forniscono un facile accesso alla risorsa coda dati.

  L'utilizzo di IBM Toolbox per Java implica che il programma opera sia sul client che sul server per accedere ad una coda dati su un server iSeries. Esso opera anche quando è in esecuzione su un server iSeries per accedere ad una coda dati su un altro server iSeries.

  L'alternativa è costituita dalla scrittura di un programma separato (in C, ad esempio) che acceda alla coda dati. Il programma Java chiama questo programma quando è necessario accedere alla coda dati.

  Questo metodo è maggiormente compatibile a livello server; è possibile avere un programma Java che gestisca l'accesso alla coda dati e versioni diverse del programma per ogni server supportato.

Esecuzione delle classi IBM Toolbox per Java sulla JVM OS/400

Le seguenti sono speciali considerazioni per l'esecuzione delle classi IBM Toolbox per Java sulla JVM di IBM Developer Kit per Java (OS/400):

Command call

Due metodi comuni di richiamare un comando sono i seguenti:

- La classe CommandCall di IBM Toolbox per Java
- Il metodo java.lang.Runtime.exec

La classe CommandCall crea un elenco di messaggi disponibili per un programma Java al completamento del comando. Tale elenco di messaggi non è disponibile tramite java.lang.Runtime.exec().

Il metodo java.lang.Runtime.exec è compatibile con molte piattaforme, pertanto, se il programma deve accedere ai file su diversi tipi di server, java.lang.Runtime.exec() costituisce la soluzione migliore.

IFS (Integrated file system)

Modi comuni di accedere ad un file nell'IFS del server iSeries:

- Le classi IFSFile del programma su licenza di IBM Toolbox per Java
- Le classi di file che fanno parte di java.io

Le classi IFS di IBM Toolbox per Java hanno il vantaggio di fornire più funzioni rispetto alle classi java.io. Le classi IBM Toolbox per Java funzionano anche nelle applet e non hanno bisogno di un metodo di reindirizzamento (come ad esempio iSeries Access per Windows) per raggiungere il server da una stazione di lavoro.

Le classi java.io sono compatibili con molte piattaforme e ciò costituisce un vantaggio. Se è necessario che il programma acceda ai file su differenti tipi di server, java.io costituisce la soluzione migliore.

Se si utilizzano le classi java.io su un client, è necessario un metodo di reindirizzamento (come ad esempio iSeries Access per Windows) per raggiungere il file system del server.

JDBC

Due unità di controllo JDBC fornite da IBM sono disponibili per i programmi che vengono eseguiti sulla JVM OS/400:

- Unità di controllo JDBC di IBM Toolbox per Java
- Unità di controllo JDBC di IBM Developer Kit per Java

L'utilizzo dell'unità di controllo JDBC di IBM Toolbox per Java è indicato quando l'unità viene eseguita in un ambiente client/server.

L'utilizzo dell'unità di controllo JDBC di IBM Developer Kit per Java è più indicato quando l'unità viene eseguita su un server iSeries.

Se lo stesso programma viene eseguito sia sulla stazione di lavoro che sul server, si dovrebbe caricare il programma di controllo corretto tramite una proprietà di sistema invece di codificare il nome del programma di controllo nel programma.

Chiamata al programma

Due modi comuni di richiamare un programma sono i seguenti:

- La classe ProgramCall di IBM Toolbox per Java
- Tramite una chiamata a JNI (Java Native Interface)

La classe ProgramCall del programma su licenza IBM Toolbox per Java possiede il vantaggio che può richiamare ogni programma del server iSeries.

L'utente potrebbe non essere in grado di richiamare il programma del server iSeries tramite JNI. JNI offre il vantaggio di essere maggiormente compatibile con le piattaforme del server.

Impostazione del nome di sistema, dell'ID utente e della parola d'ordine con un oggetto AS400 nella JVM OS/400

L'oggetto AS400 consente valori speciali per il nome di sistema, l'ID utente e la parola d'ordine quando il programma Java viene eseguito sulla JVM IBM Developer Kit per Java (OS/400).

Quando viene eseguito un programma su JVM OS/400, è necessario conoscere alcuni valori speciali ed altre considerazioni:

- La richiesta di ID utente e parola d'ordine è disattivata quando il programma è in esecuzione sul server. Per ulteriori informazioni sui valori dell'ID utente e della parola d'ordine nell'ambiente del server, consultare Riepilogo dei valori ID utente e parola d'ordine su un oggetto AS400.
- Se il nome di sistema, l'ID utente o la parola d'ordine non sono impostati nell'oggetto AS400, l'oggetto AS400 si collega al server corrente utilizzando l'ID utente e la parola d'ordine del lavoro che ha avviato il programma Java. **E' necessario fornire una parola d'ordine quando si utilizza un accesso a livello record durante il collegamento a v4r3 e da macchine con versioni precedenti. Quando avviene il collegamento ad una macchina con versione v4r4 o successive, esso può diffondere la parola d'ordine dell'utente collegato come il resto dei componenti di IBM Toolbox per Java.**
- E' possibile utilizzare il valore speciale, **localhost**, come nome di sistema.In questo caso, l'oggetto AS400 si collega al server corrente.
- E' possibile utilizzare il valore speciale, ***current**, come ID utente o parola d'ordine nell'oggetto AS400.In questo caso, vengono utilizzati l'ID utente o la parola d'ordine (o entrambi) del lavoro che ha avviato il programma Java. Per ulteriori informazioni su *current, consultare le seguenti Note.
- E' possibile utilizzare il valore speciale ***current** come ID utente o parola d'ordine nell'oggetto AS400 quando il programma Java è in esecuzione sulla JVM OS/400 di un server iSeries e il programma accede a risorse su un altro server iSeries.In questo caso, durante il collegamento al sistema di destinazione vengono utilizzati l'ID utente e la parola d'ordine del lavoro che ha avviato il programma Java nel sistema sorgente. Per ulteriori informazioni su *current, consultare le seguenti Note.

  Note:

  – Il programma Java non può impostare la parola d'ordine su "*current" se viene utilizzato l'accesso al livello record e V4R3 o versioni precedenti. Quando si utilizza l'accesso al livello record, "localhost" è valido come nome di sistema e "*current" come ID utente; tuttavia, è necessario che il programma Java fornisca la parola d'ordine.

  – *current funziona solo su sistemi che eseguono Versione 4 Release 3 (V4R3) o versioni successive. E' necessario che la parola d'ordine e l'ID utente siano specificati sul sistema in esecuzione sui sistemi V4R2.

Esempi

I seguenti esempi mostrano come utilizzare l'oggetto AS400 con JVM OS/400.

Esempio: creazione di un oggetto AS400 quando la JVM OS/400 esegue un programma Java

Quando un programma Java è in esecuzione su una JVM OS/400, il programma non deve fornire un nome di sistema, un ID utente o una parola d'ordine.

Nota: è **necessario** fornire una parola d'ordine quando si utilizza l'accesso a livello record.

Se questi valori non vengono forniti, l'oggetto AS400 si collega al sistema locale utilizzando l'ID utente e la parola d'ordine del lavoro che ha avviato il programma Java.

Quando il programma è in esecuzione sulla JVM OS/400, l'impostazione del nome di sistema su **localhost** equivale a non impostare il nome di sistema. Il seguente esempio mostra come collegarsi al server corrente:

```
// Creare due oggetti AS400. Se il programma Java è in esecuzione
     // sulla JVM OS/400, la funzionalità dei due oggetti è la stessa.
     // Essi si collegheranno al server corrente utilizzando l'ID utente e la
     // parola d'ordine del lavoro che ha avviato il programma Java.
     AS400 sys  = new AS400()
     AS400 sys2 = new AS400("localhost")
```

Esempio: collegamento al server corrente con un ID utente e una parola d'ordine differenti dal programma che ha avviato il lavoro Il programma Java può impostare un ID utente e una parola d'ordine anche se il programma è in esecuzione sulla JVM OS/400. Questi valori sostituiscono l'ID utente e la parola d'ordine del lavoro che ha avviato il programma Java.

Nell'esempio seguente, il programma Java si collega al server corrente, ma il programma utilizza un'ID utente e una parola d'ordine diversi da quelli del lavoro che ha avviato il programma Java.

```
                         // Creare un oggetto AS400.
Effettuare il collegamento al server corrente ma non utilizzare
          // l'ID utente e la parola d'ordine del lavoro che ha avviato il
                          // Java.
Vengono utilizzati i valori forniti.
          AS400 sys = new AS400("localhost", "USR2", "PSWRD2")
```

Esempio: collegamento a un server differente utilizzando l'ID utente e la parola d'ordine del lavoro che ha avviato il programma Java

Un programma Java in esecuzione su un server può collegarsi e utilizzare le risorse di altri server iSeries.

Se è utilizzato ***current** per l'ID utente e la parola d'ordine, vengono utilizzati l'ID utente e la parola d'ordine del lavoro che ha avviato il programma Java quando il programma Java si collega al server di destinazione.

Nell'esempio seguente, il programma Java viene eseguito su un server, ma utilizza le risorse di un altro server. Vengono utilizzati l'ID utente e la parola d'ordine del lavoro che ha avviato il programma Java quando il programma si collega al secondo server.

```
                         // Creare un oggetto AS400.
Questo programma verrà eseguito su un server
     // ma si collegherà ad un secondo server (detto "destinazione").
     // Poiché viene utilizzato *current per usare l'ID utente e la parola d'ordine,
     // verranno utilizzati l'ID utente e la parola d'ordine del lavoro che ha avviato il
     // programma quando ci si collega al secondo server.
          AS400 target = new AS400("target", "*current", "*current")
```

Riepilogo dei valori ID utente e parola d'ordine su un oggetto AS400:

La tabella seguente riepiloga le modalità di gestione dell'ID utente e della parola d'ordine su un oggetto AS400 da parte di un programma Java in esecuzione su un server a confronto con un programma Java in esecuzione su un client.

<table>
<tr><th>Valori sull'oggetto AS400</th><th>Programma Java in esecuzione su un server</th><th>Programma Java in esecuzione su un client</th></tr>
<tr><td>Nome sistema, ID utente e parola d'ordine non impostati</td><td>Collegarsi al server corrente utilizzando l'ID utente e la parola d'ordine del lavoro che ha avviato il programma</td><td>Richiede il sistema, l'ID utente e la parola d'ordine</td></tr>
<tr><td>Nome sistema = localhost</td><td rowspan="3">Collegarsi al server corrente utilizzando l'ID utente e la parola d'ordine del lavoro che ha avviato il programma</td><td rowspan="3">Errore: localhost non è valido quando il programma Java è in esecuzione su un client</td></tr>
<tr><td>Nome sistema = localhost ID utente = *current</td></tr>
<tr><td>Nome sistema = localhost ID utente = *current ID parola d'ordine = *current</td></tr>
<tr><td>Nome sistema = ″sys″</td><td>Collegarsi al server ″sys″ utilizzando l'ID utente e la parola d'ordine del lavoro che ha avviato il programma. ″sys″ può essere il server corrente o un altro server</td><td>Richiedere l'ID utente e la parola d'ordine</td></tr>
<tr><td>Nome sistema = localhost ID utente = ″UID″ ID parola d'ordine = ″PWD″</td><td>Collegarsi al server corrente utilizzando l'ID utente e la parola d'ordine specificati dal programma Java invece dell'ID utente e della parola d'ordine del lavoro che ha avviato il programma</td><td>Errore: localhost non è valido quando il programma Java non è in esecuzione su un client</td></tr>
</table>

IASP (Independent auxiliary storage pool)

IASP (independent auxiliary storage pool) è un insieme di unità disco che è possibile portare in linea o non in linea indipendentemente dalla quantità di memoria rimasta su un sistema. IASP contiene quanto segue:

- uno o più file system definiti dall'utente
- uno o più archivi esterni

Ogni IASP contiene tutte le informazioni necessarie per un sistema e i dati del sistema stesso. Quindi, mentre il sistema è attivo, è possibile scollegare l'IASP, collegarlo nuovamente o effettuare uno switch tra i sistemi.

Per ulteriori informazioni, consultare IASP e ASP utente.

E' possibile utilizzare la proprietà JDBC ″database name″ o il metodo setDatabaseName() dalla classe AS400JDBCDataSource per specificare l'ASP a cui si desidera collegarsi.

Tutte le altre classi Toolbox per Java (IFSFile, Print, DataQueues e altre) utilizzano l'IASP specificato dalla descrizione del lavoro del profilo utente che si collega al server.

Ottimizzazione di OS/400

Il programma su licenza IBM Toolbox per Java è scritto in Java, quindi esso viene eseguito su tutte le piattaforme con una JVM (Java virtual machine) certificata. Le classi IBM Toolbox per Java funzionano nello stesso modo senza tener conto dell'ubicazione in cui si eseguono.

Classi supplementari vengono fornite con OS/400, le quali migliorano il funzionamento di IBM Toolbox per Java quando si esegue sulla JVM di iSeries. Il funzionamento in fase di collegamento e le prestazioni

vengono migliorati durante l'esecuzione sulla JVM di iSeries e il collegamento allo stesso server iSeries. OS/400 ha incorporato le classi supplementari a partire dalla Versione 4 Rilascio 3.

Abilitazione delle ottimizzazioni

IBM Toolbox per Java viene consegnato in due pacchetti: come programma su licenza separato e con OS/400.

- Programma su licenza 5722-JC1. La versione programma su licenza di IBM Toolbox per Java invia i file nel seguente indirizzario:

  ```
  /QIBM/ProdData/http/public/jt400/lib
  ```

  Questi file non contengono le ottimizzazioni OS/400. Utilizzare questi file se si desidera un funzionamento coerente con l'esecuzione di IBM Toolbox per Java su un client.
- OS/400. IBM Toolbox per Java viene anche trasportato nell'indirizzario con OS/400

  ```
  /QIBM/ProdData/OS400/jt400/lib
  ```

  Questi file contengono le classi che ottimizzano IBM Toolbox per Java durante l'esecuzione sulla JVM iSeries.

  Per ulteriori informazioni consultare Nota 1 nelle informazioni sui file Jar.

Considerazioni sul collegamento

Con le classi aggiuntive fornite con OS/400, i programmi Java hanno opzioni aggiuntive per fornire informazioni sul nome del server (sistema), sull'ID utente e sulla parola d'ordine ad IBM Toolbox per Java.

Durante l'accesso ad una risorsa su un server iSeries, le classi IBM Toolbox per Java devono disporre di un nome di sistema, di un ID utente e di una parola d'ordine.

- **Durante l'esecuzione su un client**, il nome di sistema, l'ID utente e la parola d'ordine vengono forniti dal programma Java o IBM Toolbox per Java richiama questi valori dall'utente tramite una finestra di dialogo di collegamento.
- **Durante l'esecuzione sulla JVM iSeries**, l'IBM Toolbox per Java ha una opzione in più. Esso può inviare richieste al server corrente (locale) utilizzando l'ID utente e la parola d'ordine del lavoro che ha avviato il programma Java.

Con le classi aggiuntive, l'ID utente e la parola d'ordine del lavoro corrente possono essere utilizzati anche quando un programma Java in esecuzione su un server iSeries accede alle risorse su un altro server iSeries. In questo caso, il programma Java imposta il nome di sistema, quindi utilizza il valore speciale "*current" per l'ID utente e la parola d'ordine.

Il programma Java può impostare la parola d'ordine solo su "*current" se si sta utilizzando V4R4 o versioni successive in accesso al livello record. Altrimenti, quando si utilizza l'accesso al livello record, "localhost" è valido per il nome di sistema e "*current" è valido per l'ID utente; tuttavia, il programma Java deve fornire la parola d'ordine.

Un programma Java imposta i valori del nome di sistema, l'ID utente e della parola d'ordine nell'oggetto AS400 .

Per utilizzare l'ID utente e la parola d'ordine del lavoro, il programma Java può utilizzare "*current" come ID utente e parola d'ordine oppure può utilizzare il programma di creazione che non possiede i parametri ID utente e parola d'ordine.

Per utilizzare l'iSeries corrente, il programma Java può utilizzare "localhost" come nome del sistema o utilizzare il programma di creazione predefinito. Cioè,

```
AS400 system = new AS400();
```

è uguale a

```
AS400 system = new AS400("localhost", "*current", "*current");
```

Esempi

I seguenti esempi mostrano come collegarsi al server utilizzando le classi ottimizzate.

Esempio: come collegarsi quando si utilizzando programmi di creazione AS400 differenti

Nel seguente esempio vengono creati due oggetti AS400. I due oggetti hanno la stessa funzione: entrambi eseguono un comando sul server corrente utilizzando l'ID utente e la parola d'ordine del lavoro. Un oggetto utilizza il valore speciale per l'ID utente e la parola d'ordine, mentre l'altro utilizza il programma di creazione predefinito e non imposta l'ID utente o la parola d'ordine.

```
                    // Creare un oggetto AS400.
Poiché il programma
                    // di creazione predefinito e l'ID utente e la parola d'ordine
                    // e il sistema non sono mai stati impostati, l'oggetto AS400 invia
                    // all'iSeries locale utilizzando l'ID utente e la parola
                    // del lavoro. Se questo programma è stato eseguito
                    // su un client, all'utente verrà richiesto l'ID utente, la parola
                    // d'ordine e il sistema.
    AS400 sys1 = new AS400();

                    // Creare un oggetto AS400.
Tale oggetto invia richieste
                    // all'iSeries locale utilizzando l'ID utente e la parola
                    // del lavoro. Tale oggetto non funzionerà su
                    // un client.
    AS400 sys2 = new AS400("localhost", "*current", "*current");

                    // Creare due oggetti di chiamata al comando che utilizzano
                    // gli oggetti AS400.
    CommandCall cmd1 = new CommandCall(sys1,"myCommand1");
    CommandCall cmd2 = new CommandCall(sys2,"myCommand2");

                    // Eseguire i comandi.
    cmd1.run();
    cmd2.run();
```

Esempio: collegarsi utilizzando l'ID utente e la parola d'ordine del lavoro corrente

Nel seguente esempio viene creato un oggetto AS400 che rappresenta un secondo server iSeries. Poiché si utilizza "*current", l'ID utente e la parola d'ordine del lavoro di un server iSeries su cui è in esecuzione un programma Java vengono utilizzati nel secondo server (di destinazione).

```
                    // Creare un oggetto AS400.
Tale oggetto invia richieste
                    // ad un secondo iSeries utilizzando l'ID utente e la
                    // parola d'ordine del lavoro sul server corrente.
    AS400 sys = new AS400("mySystem.myCompany.com", "*current", "*current");


                    // Creare un oggetto chiamata al comando per eseguire un comando
                    // sul server di destinazione.
    CommandCall cmd = new CommandCall(sys,"myCommand1");


                    // Eseguire il comando.
    cmd.run();
```

Miglioramenti delle prestazioni

Con le classi aggiuntive fornite da OS/400, i programmi Java in esecuzione sulla JVM per iSeries si avvalgono di prestazioni potenziate. Le prestazioni vengono migliorate in alcuni casi poiché si utilizza in minore misura la funzione di comunicazione e, in altri casi, si utilizza un'API iSeries invece di richiamare il programma del server.

Minore tempo di scaricamento

Al fine di scaricare il numero minimo di file classe IBM Toolbox per Java, utilizzare il server proxy insieme allo strumento AS400ToolboxJarMaker.

Comunicazione più rapida

Per tutte le funzioni di IBM Toolbox per Java ad eccezione di JDBC e dell'accesso IFS, i programmi Java in esecuzione sulla JVM per iSeries saranno più veloci. I programmi sono più veloci perché si utilizza in minore misura il codice di comunicazione quando si comunica tra il programma Java e il programma del server sul server che effettua la richiesta.

JDBC e l'accesso IFS non sono stati ottimizzati poiché esistono già applicazioni che rendono l'esecuzione di tali funzioni più veloce. Quando è in esecuzione sull'iSeries, è possibile utilizzare l'unità di controllo JDBC per iSeries invece dell'unità di controllo JDBC fornita con IBM Toolbox per Java. Per accedere ai file sul server, è possibile utilizzare java.io invece delle classi di accesso IFS in dotazione con IBM Toolbox per Java.

Chiamata diretta alle API OS/400

Le prestazioni delle seguenti classi di IBM Toolbox per Java risultano potenziate poiché queste classi chiamano direttamente le API OS/400 invece di chiamare un programma server per eseguire la richiesta:

- Classi AS400Certificate
- CommandCall
- DataQueue
- ProgramCall
- Classi di accesso al database a livello record
- ServiceProgramCall
- UserSpace

Le API vengono direttamente richiamate solo se l'ID utente e la parola d'ordine corrispondono all'ID utente e alla parola d'ordine del lavoro che esegue il programma Java. Per ottenere un miglioramento nelle prestazioni, è necessario che l'ID utente e la parola d'ordine corrispondano all'ID utente e alla parola d'ordine del lavoro che avvia il programma Java. Per realizzare risultati migliori, utilizzare ″localhost″ per il nome di sistema, ″*current″ per l'ID utente e ″*current″ per la parola d'ordine.

Modifiche alla definizione porta

Il sistema di definizione porta è stato modificato, rendendo più veloce l'accesso ad una porta. Prima di tale modifica, una richiesta per una porta sarebbe stata inviata al programma di definizione porta. A quel punto, il server iSeries avrebbe stabilito quale porta fosse disponibile e l'avrebbe restituita all'utente per l'accettazione. Ora, è possibile comunicare al server quale porta utilizzare o specificare che devono essere utilizzate le porte predefinite. Questa opzione consente di eliminare inutili sprechi di tempo dovuti alla determinazione della porta da parte del server per conto dell'utente. Utilizzare il comando WRKSRVTBLE per visualizzare o modificare l'elenco di porte per il server.

Per un miglioramento nella definizione porta, è stato aggiunto qualche metodo alla classe AS400:

- getServicePort
- setServicePort
- setServicePortsToDefault

Modifiche di stringhe specifiche della lingua

I file di stringhe specifiche della lingua ora vengono forniti nel programma di IBM Toolbox per Java come file di classe invece che come file di proprietà. Il server iSeries individua più rapidamente messaggi nei file di classe che nei file di proprietà. ResourceBundle.getString() ora viene eseguito più velocemente poiché i file sono memorizzati nella prima ubicazione su cui il computer effettua la ricerca. Un altro vantaggio della modifica in file di classe risiede nella maggiore velocità con cui il server può trovare la versione tradotta di una stringa.

Programmi di conversione

Due classi consentono una conversione più rapida e più efficiente tra Java e iSeries:

- Programma di conversione binario: effettua la conversione tra le schiere di byte Java e i tipi semplici Java.
- Programma di conversione carattere: effettua la conversione tra oggetti Java String e code page OS/400.

Inoltre, IBM Toolbox per Java contiene ora le proprie tabelle di conversione per oltre 100 CCSID comunemente utilizzati. In precedenza, IBM Toolbox per Java si rimetteva a Java per quasi tutta la conversione testo. Se Java non possedeva la tabella di conversione corretta, IBM Toolbox per Java scaricava la tabella di conversione dal server.

IBM Toolbox per Java esegue tutta la conversione testo per ogni CCSID riconosciuto. Quando incontra un CCSID non riconosciuto, tende ad affidare a Java la gestione della conversione. In nessun caso IBM Toolbox per Java cerca di scaricare una tabella di conversione dal server. Questa tecnica riduce di gran lunga il tempo impiegato da un applicazione di IBM Toolbox per Java per eseguire la conversione del testo. Non si richiede alcuna operazione da parte dell'utente per trarre vantaggio dalla nuova conversione del testo; i miglioramenti delle prestazioni si verificano tutti nelle tabelle del programma di conversione sottostante.

Suggerimento sulle prestazioni riguardanti il comando CRTJVAPGM (Creazione programma Java)

Se l'applicazione Java si esegue sulla JVM (Java virtual machine) iSeries, è possibile **migliorare le prestazioni in modo significativo** se si crea un programma Java da un file .zip o .jar di IBM Toolbox per Java. Immettere il comando **CRTJVAPGM** su una riga comandi iSeries per creare il programma. (Consultare le informazioni dell'aiuto in linea per il comando **CRTJVAPGM** per ulteriori approfondimenti.) Utilizzando il comando **CRTJVAPGM**, viene salvato il programma Java creato (e contenente le classi IBM Toolbox per Java), quando si avvia la propria applicazione Java. Il salvataggio del programma Java creato consente di salvare il tempo di elaborazione dell'avvio. Il tempo di elaborazione dell'avvio viene salvato poiché non è necessario ricreare il programma Java sul server ogni volta che viene avviata la propria applicazione Java.

Se si utilizza la versione V4R2 o V4R3 di IBM Toolbox per Java, è impossibile eseguire il comando **CRTJVAPGM** sul file jt400.zip o jt400.jar perché è troppo grande; tuttavia, è forse possibile eseguirlo sul file jt400Access.zip. Nella V4R3, il programma su licenza IBM Toolbox per Java comprende un file aggiuntivo, jt400Access.zip. jt400Access.zip contiene solo le classi di accesso, non le classi visuali.

Quando si eseguono le applicazioni Java su un sistema V4R5 (o precedente) utilizzare jt400Access.zip. Quando si eseguono le applicazioni Java su un sistema V5R1, utilizzare jt400Native.jar. Il comando **CRTJVAPGM** è stato già eseguito su jt400Native.jar.

National language support di Java

Java supporta una serie di linguaggi nazionali, ma si tratta si una sottoserie dei linguaggi supportati dal server.

Quando si verifica una mancata corrispondenza tra lingue, ad esempio, se si sta effettuando una esecuzione su una stazione di lavoro locale che utilizza una lingua non supportata da Java, il programma su licenza di IBM Toolbox per Java **può emettere alcuni messaggi di errore in Inglese**.

Servizio e supporto per IBM Toolbox per Java

Utilizzare le risorse che seguono per il servizio e il supporto:

Informazioni per la risoluzione dei problemi di IBM Toolbox per Java Utilizzare queste informazioni come aiuto per la risoluzione di problemi quando si utilizza IBM Toolbox per Java.

Forum JTOpen/IBM Toolbox per Java partecipare alla comunità di programmatori Java che utilizzano IBM Toolbox per Java. Questo forum è un modo efficace per ottenere assistenza e suggerimenti da altri programmatori Java e talvolta dagli stessi sviluppatori di IBM Toolbox per Java

Supporto server Utilizzare il sito web del Supporto server IBM per acquisire informazioni sugli strumenti e le risorse utili a snellire la pianificazione tecnica ed il supporto per il server iSeries.

Supporto software Utilizzare il sito Web Servizi di supporto software IBM per acquisire informazioni su una vasta gamma di servizi di supporto software offerti da IBM.

I servizi di supporto per IBM Toolbox per Java, 5722-JC1, sono forniti sotto gli stessi termini e condizioni dei prodotti software iSeries. I servizi di supporto includono servizi di programma, supporto a voce e servizi di consultazione. Per ulteriori informazioni contattare il rappresentante IBM locale.

La risoluzione dei difetti del programma IBM Toolbox per Java è supportata sotto i servizi di programma e il supporto a voce, mentre la risoluzione dei problemi della programmazione e dell'esecuzione del debug dell'applicazione è supportato sotto i servizi di consultazione.

Le chiamate API di IBM Toolbox per Java sono supportate sotto i servizi di consultazione affinché almeno uno di quanto segue sia vero:

- E' chiaramente un difetto API Java, dimostrato da una nuova creazione in un programma relativamente semplice.
- E' una questione di richiesta della chiarificazione della documentazione.
- E' una questione relativa alla posizione dei campioni o della documentazione.

Tutta l'assistenza di programmazione viene supportata sotto i servizi di consultazione inclusi i programmi campione forniti nel programma su licenza IBM Toolbox per Java. Campioni aggiuntivi possono essere disponibili su Internet sulla pagina iniziale di iSeries su una base non supportata.

Le informazioni di risoluzione problema sono fornite con il prodotto programma su licenza IBM Toolbox per Java. Se l'utente crede che esista un potenziale difetto nell'API IBM Toolbox per Java API, sarà necessario un semplice programma che dimostra l'errore.

Esempi di codice

Il seguente elenco fornisce collegamenti ai punti di immissione per molti degli esempi utilizzati in tutte le informazioni di IBM Toolbox per Java.

Classi Access
Bean
Classi Commtrace
Graphical Toolbox
Classi HTML
PCML
Classi ReportWriter
Classi Resource
RFML
Classi Security
Classi Servlet
Esempi semplici
Suggerimenti per la programmazione
ToolboxMe per iSeries
Classi programma di utilità
Classi Vaccess
XPCML

Il seguente esonero di responsabilità si applica a tutti gli esempi IBM Toolbox per Java:

Esonero di responsabilità per gli esempi di codice

IBM fornisce una licenza non esclusiva per utilizzare ciò come esempio da cui creare funzioni simili personalizzate, in base a richieste specifiche.

Tutti gli esempi di codice forniti dall'IBM hanno la sola funzione illustrativa. Questi esempi non sono stati interamente testati in tutte le condizioni. L'IBM, perciò, non intende implicita alcuna garanzia di affidabilità, manutenibilità o funzionalità di questi programmi.

Tutti i programmi qui contenuti sono forniti ″COSI' COME SONO″ senza garanzie di alcun tipo. Sono espressamente smentite tutte le garanzie implicite di non violazione, di commerciabilità e idoneità per scopi specifici.

Esempi: classi Access

Questa sezione elenca gli esempi di codice forniti in tutta la documentazione delle classi Access di IBM Toolbox per Java.

AS400JPing

- Esempio: utilizzo di AS400JPing in un programma Java

BidiTransform

- Esempio: utilizzo della classe AS400BidiTransform per convertire testo bidirezionale

CommandCall

- Esempio: utilizzo di CommandCall per eseguire un comando sul server
- Esempio: utilizzo di CommandCall per richiedere il nome del server, il comando per eseguire e stampare il risultato

ConnectionPool

- Esempio: utilizzo di AS400ConnectionPool per creare i collegamenti al server

DataArea

- Esempio: creazione e scrittura in un'area dati decimali

DataConversion e DataDescription

- Esempi: utilizzo delle classi FieldDescription, RecordFormat e Record
- Esempio: immissione di dati in una coda
- Esempio: lettura di dati da una coda
- Esempio: utilizzo delle classi AS400DataType con ProgramCall

DataQueue

- Esempio: come creare un oggetto DataQueue, leggere dati e scollegarsi
- Esempio: immissione di dati in una coda
- Esempio: lettura di dati da una coda
- Esempio: utilizzo di KeyedDataQueue per inserire elementi a una coda
- Esempio: utilizzo di KeyedDataQueue per recuperare elementi da una coda

Certificato digitale

- Esempio: elenco dei certificati digitali che appartengono ad un utente

EnvironmentVariable

- Esempio: creazione, impostazione e richiamo di variabili di ambiente

Eccezioni

- Esempio: come catturare un'eccezione inviata, richiamare un codice di ritorno e visualizzare il testo dell'eccezione

FTP

- Esempio: utilizzo della classe FTP per copiare una serie di file da un indirizzario del server
- Esempio: utilizzo della classe AS400FTP per copiare una serie di file da un indirizzario

IFS (Integrated file system)

- Esempi: utilizzo di IFSFile
- Esempio: utilizzo del metodo IFSFile.listFiles() per elencare il contenuto di un indirizzario
- Esempio: utilizzo delle classi IFSFile per copiare i file
- Esempio: utilizzo delle classi IFSFile per elencare il contenuto di un indirizzario
- Esempio: come utilizzo di IFSJavaFile invece di java.io.File
- Esempio: utilizzo delle classi IFSFile per elencare il contenuto di un indirizzario nel server

JavaApplicationCall

- Esempio: esecuzione di un programma nel server dal client che emette ″Hello World!″

JDBC

- Esempio: utilizzo dell'unità di controllo JDBC per creare e popolare una tabella
- Esempio: utilizzo dell'unità di controllo JDBC per interrogare una tabella ed emettere il suo contenuto

Lavori

- Esempio: richiamo e modifica delle informazioni di lavoro utilizzando la memoria cache
- Esempio: elenco di tutti i lavori attivi
- Esempio: stampa di tutti i messaggi della registrazione lavori per un determinato utente
- Esempio: elenco delle informazioni di identificazione del lavoro per un determinato utente
- Esempio: richiamo di un elenco di lavori nel server ed elencare lo stato del lavoro ed il relativo ID

- Esempio: visualizzazione dei messaggi nella registrazione lavori per un lavoro che appartiene all'utente corrente

Coda messaggi
- Esempio: come utilizzare l'oggetto coda messaggi
- Esempio: stampa del contenuto di una coda messaggi
- Esempio: richiamo e stampa di un messaggio
- Esempio: elenco del contenuto della coda messaggi
- Esempio: utilizzo di AS400Message con CommandCall
- Esempio: utilizzo di AS400Message con ProgramCall

NetServer
- Esempio: utilizzo di un oggetto NetServer per modificare il nome del NetServer

Stampa
- Esempio: creazione di un elenco asincrono di tutti i file di spool utilizzando l'interfaccia PrintObjectListListener
- Esempio: creazione di un elenco asincrono di tutti i file di spool *senza* utilizzare l'interfaccia PrintObjectListListener
- Esempio: copia di un file di spool utilizzando SpooledFile.copy()
- Esempio: creazione di un file di spool da un flusso di immissione
- Esempio: creazione di un flusso di dati SCS utilizzando la classe SCS3812Writer
- Esempio: lettura di un file di spool esistente
- Esempio: lettura e trasformazione dei file di spool
- Esempio: creazione di un elenco asincrono di tutti i file di spool

Autorizzazione
- Esempio: impostazione dell'autorizzazione di un oggetto AS400

Chiamata al programma
- Esempio: utilizzo di ProgramCall
- Esempio: utilizzo di ProgramCall per richiamare lo stato del sistema
- Esempio: inoltro di dati del parametro con un oggetto Program parameter

QSYSObjectPathName
- Esempio: creazione di un nome IFS
- Esempio: utilizzo di QSYSObjectPathName.toPath() per creare un nome oggetto di AS400
- Esempio: utilizzo di QSYSObjectPathName per analizzare il nome del percorso IFS

Accesso al livello record
- Esempio: accesso sequenziale ad un file
- Esempio: utilizzo delle classi di accesso al livello record per leggere un file
- Esempio: utilizzo delle classi di accesso al livello record per leggere i record tramite chiave
- Esempio: utilizzo della classe LineDataRecordWriter

Chiamata al programma di servizio
- Esempio: utilizzo di ServiceProgramCall per richiamare una procedura

SystemStatus
- Esempio: utilizzo della memorizzazione in cache con la classe SystemStatus

SystemPool
- Esempio: impostazione della dimensione massima degli errori per SystemPool

SystemValue
- Esempio: utilizzo di SystemValue e SystemValueList

Traccia
- Esempio: utilizzo del metodo Trace.setTraceOn()
- Esempio: miglior utilizzo di Trace
- Esempio: utilizzo della traccia del componente

UserGroup
- Esempio: richiamo di un elenco di utenti
- Esempio: creazione di un elenco di tutti gli utenti di un gruppo

UserSpace
- Esempio: come creare uno spazio utente

Il seguente esonero di responsabilità si applica a tutti gli esempi IBM Toolbox per Java:

Esempio: utilizzo di CommandCall

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per informazioni legali importanti.

```
////////////////////////////////////////////////////////////////////////////
//
// Esempio CommandCall. Questo programma richiede all'utente
// il nome del server ed il comando da eseguire, quindi
// stampa il risultato del comando.
//
// Questo sorgente, è un esempio di "CommandCall" di IBM Toolbox per Java
//
////////////////////////////////////////////////////////////////////////////



import java.io.*;
import java.util.*;
import com.ibm.as400.access.*;

public class CommandCallExample extends Object
{
```

```
   public static void main(String[] parmeters)
   {

      // Creato un programma di lettura per richiamare l'immissione dall'utente
      BufferedReader inputStream = new BufferedReader(new InputStreamReader(System.in),1);



      // Dichiarare le variabili per contenere il nome di sistema ed il comando da eseguire
      String systemString  = null;
      String commandString = null;

      System.out.println( " " );



      // Richiamare il nome di sistema ed il comando da eseguire dall'utente
      try
      {
         System.out.print("System name: ");
         systemString = inputStream.readLine();

         System.out.print("Command: ");
         commandString = inputStream.readLine();
      }
      catch (Exception e) {};

      System.out.println( " " );



      // Creare un oggetto AS400.
Questo è il sistema a cui viene inviato il comando
      AS400 as400 = new AS400(systemString);



      // Creare un oggetto chiamata al comando specificando il server che
      // riceverà il comando.
      CommandCall command = new CommandCall( as400 );



      try
      {
         // Eseguire il comando.
         if (command.run(commandString))
             System.out.print( "Command successful" );
         else
             System.out.print( "Command failed" );



         // Se vengono prodotti messaggi dal comando, occorre stamparli
         AS400Message[] messagelist = command.getMessageList();

         if (messagelist.length > 0)
         {
             System.out.println( ", messages from the command:" );
             System.out.println( " " );
         }

         for (int i=0; i < messagelist.length; i++)
         {
            System.out.print  ( messagelist[i].getID() );
            System.out.print  ( ": " );
            System.out.println( messagelist[i].getText() );
```

```
         }
      }
      catch (Exception e)
      {
         System.out.println( "Command " + command.getCommand() + " did not run" );
      }

      System.exit(0);
   }
}
```

Esempio: utilizzo di AS400ConnectionPool

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per informazioni legali importanti.

```
////////////////////////////////////////////////////////////////////////////////
//
// Esempio AS400ConnectionPooling. Questo programma utilizza un AS400ConnectionPool
// per creare i collegamenti a un server iSeries.
// Sintassi del comando:
// AS400ConnectionPooling system myUserId myPassword
//
// Ad esempio,
// AS400ConnectionPooling MySystem MyUserId MyPassword
//
////////////////////////////////////////////////////////////////////////////////

import com.ibm.as400.access.*;

public class AS400ConnectionPooling
{
      public static void main (String[] parameters)
   {
            // Controllare i parametri di immissione.
            if (parameters.length != 3)
      {
         System.out.println("");
                  System.out.println("Usage:");
         System.out.println("");
                  System.out.println("   AS400ConnectionPooling system userId password");
         System.out.println("");
         System.out.println("");
         System.out.println("For example:");
         System.out.println("");
         System.out.println("");
                  System.out.println("   AS400ConnectionPooling MySystem MyUserId MyPassword");
         System.out.println("");
         return;
      }

               String system = parameters[0];
         String userId   = parameters[1];
         String password = parameters[2];

                                 try
      {
         // Creare un AS400ConnectionPool.
                  AS400ConnectionPool testPool = new AS400ConnectionPool();

                  // Impostare un massimo di 128 collegamenti per questo lotto.
                  testPool.setMaxConnections(128);

                  // Impostare una durata massima di 30 minuti per i collegamenti.
                  testPool.setMaxLifetime(1000*60*30);     // Durata massima 30 min dalla creazione.

                  // Precollegare 5 collegamenti al servizio AS400.COMMAND.
                  testPool.fill(system, userId, password, AS400.COMMAND, 1);
```

```
        System.out.println();
             System.out.println("Preconnected 1 connection to the AS400.COMMAND service");

             // Chiamare getActiveConnectionCount e getAvailableConnectionCount per verificare quanti
             // collegamenti sono in uso o disponibili per uno specifico sistema.
   System.out.println("Number of active connections: "
                      + testPool.getActiveConnectionCount(system, userId));
   System.out.println("Number of available connections for use: "
                      + testPool.getAvailableConnectionCount(system, userId));

   // Creare un collegamento al servizio AS400.COMMAND. (Utilizzare le costanti del numero
   // servizio definite nella classe AS400 (FILE, PRINT, COMMAND, DATAQUEUE e così via.))
   // Poiché i collegamenti sono già stati stabiliti, viene risparmiato il tempo normalmente richiesto
   // per collegarsi al servizio comando.
             AS400 newConn1 = testPool.getConnection(system, userId, password, AS400.COMMAND);

        System.out.println();
             System.out.println("getConnection gives out an existing connection to user");
   System.out.println("Number of active connections: "
                      + testPool.getActiveConnectionCount(system, userId));
   System.out.println("Number of available connections for use:  "
                      + testPool.getAvailableConnectionCount(system, userId));

             // Creare un nuovo oggetto chiamata al comando ed eseguire un comando.
             CommandCall cmd1 = new CommandCall(newConn1);
             cmd1.run("CRTLIB FRED");

             // Restituire il collegamento al lotto.
             testPool.returnConnectionToPool(newConn1);

        System.out.println();
             System.out.println("Returned a connection to pool");
   System.out.println("Number of active connections: "
                      + testPool.getActiveConnectionCount(system, userId));
   System.out.println("Number of available connections for reuse: "
                      + testPool.getAvailableConnectionCount(system, userId));

   // Creare un collegamento al servizio AS400.COMMAND. Questa operazione restituirà lo stesso
             // oggetto riportato sopra per un nuovo utilizzo.
             AS400 newConn2 = testPool.getConnection(system, userId, password, AS400.COMMAND);

        System.out.println();
             System.out.println("getConnection gives out an existing connection to user");
   System.out.println("Number of active connections: "
                      + testPool.getActiveConnectionCount(system, userId));
   System.out.println("Number of available connections for reuse: "
                      + testPool.getAvailableConnectionCount(system, userId));

   // Creare un collegamento al servizio AS400.COMMAND. Questa operazione creerà un nuovo
             // collegamento poiché non vi sono collegamenti nel lotto da riutilizzare.
             AS400 newConn3 = testPool.getConnection(system, userId, password, AS400.COMMAND);

        System.out.println();
   System.out.println("getConnection creates a new connection because there are no
                      connections available");
   System.out.println("Number of active connections: "
                      + testPool.getActiveConnectionCount(system, userId));
   System.out.println("Number of available connections for reuse: "
                      + testPool.getAvailableConnectionCount(system, userId));

             // Chiudere il lotto di verifica.
             testPool.close();
}
                             catch (Exception e)
{
   // Se una qualsiasi delle operazioni elencate in precedenza ha dato esito
   //negativo considerare non riuscite le operazioni del lotto
```

```
        // ed emettere l'eccezione.

                 System.out.println("Pool operations failed");
                System.out.println(e);
                e.printStackTrace();
      }
   }
}
```

Esempi: utilizzo delle classi FieldDescription, RecordFormat e Record

I seguenti esempi mostrano il modo in cui è possibile utilizzare le classi FieldDescription, RecordFormat e Record con code dati.

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

Esempio: utilizzo delle classi FieldDescription

E' possibile utilizzare le classi FieldDescription per descrivere i diversi tipi di dati che costituiscono un'immissione sulla coda dati. Questi esempi assumono il seguente formato per le voci sulla coda dati:

```
   Message number    Sender    Time sent      Message text     Reply required
       |               |           |              |                |
         bin(4)        char(50)    char(8)         char(1024)        char(1)


   // Creare le descrizioni campo per i dati di immissione
 BinaryFieldDescription msgNumber = new BinaryFieldDescription(new AS400Bin4(),
                                    "msgnum");
 CharacterFieldDescription sender = new CharacterFieldDescription(new AS400Text(50),
                                    "sender");
 CharacterFieldDescription timeSent = new CharacterFieldDescription(new AS400Text(8),
                                      "timesent");
 CharacterFieldDescription msgText = new CharacterFieldDescription(new AS400Text(1024),
                                     "msgtext");
 CharacterFieldDescription replyRequired = new CharacterFieldDescription(new AS400Text(1),
                                           "replyreq");
```

Utilizzo della classe RecordFormat

E' possibile utilizzare la classe RecordFormat per descrivere i dati che costituiscono l'immissione dei dati in coda.

Esempio: definizione di RecordFormat e relativo utilizzo dinamicamente

Il seguente esempio utilizza la classe RecordFormat per descrivere il formato per la voce sulla coda dati e successivamente lo utilizza dinamicamente per richiamare un record:

```
RecordFormat entryFormat = new RecordFormat();
// Descrivere i campi in una immissione sulla coda dati
entryFormat.addFieldDescription(msgNumber);
entryFormat.addFieldDescription(sender);
entryFormat.addFieldDescription(timeSent);
entryFormat.addFieldDescription(msgText);
entryFormat.addFieldDescription(replyRequired);

// Richiamare un record in base al formato delle voci della coda dati
Record rec = entryFormat.getNewRecord();
```

Esempio: definizione statica di RecordFormat

Il seguente esempio utilizza il formato record staticamente, consentendo a diversi programmi di utilizzare il formato senza codificare il formato record più volte.

```
  public class MessageEntryFormat extends RecordFormat
{
      // Le descrizioni campi si trovano nella classe
  static BinaryFieldDescription msgNumber = new BinaryFieldDescription(new AS400Bin4(),
                                          "msgnum");
  static CharacterFieldDescription sender = new CharacterFieldDescription(new AS400Text(50),
                                          "sender");
  static CharacterFieldDescription timeSent = new CharacterFieldDescription(new AS400Text(8),
                                           "timesent");
  static CharacterFieldDescription msgText = new CharacterFieldDescription(new AS400Text(1024),
                                           "msgtext");
  static CharacterFieldDescription replyRequired = new CharacterFieldDescription(new AS400Text(1),
                                                 "replyreq");

      public MessageEntryFormat()
  {
          // Denomineremo questo formato per il futuro
    super("MessageEntryFormat");
          // Aggiungere le descrizioni campo
          addFieldDescription(msgNumber);
          addFieldDescription(sender);
          addFieldDescription(timeSent);
          addFieldDescription(msgText);
          addFieldDescription(replyRequired);
  }
}
```

Esempio: utilizzo statico di RecordFormat

Il seguente esempio mostra il modo in cui un programma Java può utilizzare staticamente un RecordFormat:

```
MessageEntryFormat entryFormat = new MessageEntryFormat();
// Richiamare un record in base al formato delle voci della coda dati
Record rec = entryFormat.getNewRecord();
```

Utilizzo della classe Record

E' possibile utilizzare la classe Record per accedere a singoli campi di voci sulla coda dati.

Esempio: utilizzo di un oggetto Record generico

```
  // Avviare l'oggetto della coda dati
DataQueue dq = new DataQueue(new AS400(), "/qsys.lib/mylib.lib/myq.dtaq");

  // Leggere una voce
  DataQueueEntry dqEntry = null;
                           try
{
      dqEntry = dq.read();
}
  catch(Exception e)
{
      // Gestire qualsiasi eccezione
}

  // Richiamare un oggetto record dal formato record, inizializzarlo con i dati che derivano dalla
  // voce appena letta.
  Record rec = entryFormat.getNewRecord(dqEntry.getData());

  // Emettere la voce completa come String. Il contenuto del record viene convertito in Oggetti Java
// basati sul formato record della voce.
  System.out.println(rec.toString());
  // Richiamare il contenuto dei singoli campi della voce. Ogni contenuto del campo viene convertito
// in un oggetto Java.
  Integer num = (Integer)rec.getField(0);    // Reperire il contenuto per indice
```

```
String s = (String)rec.getField("sender");// Reperire il contenuto per nome campo
  String text = (String)rec.getField(3);    // Reperire il testo del messaggio
  // Emettere i dati
System.out.println(num + " " + s + " " + text);
```

Esempio: utilizzo di un oggetto Record specifico

E' possibile anche definire staticamente e utilizzare un Record specifico per il formato di questa coda dati, consentendo di fornire i metodi get() e set() per i campi più significativamente denominati getField() e setField(). Inoltre, utilizzando il Record specifico definito staticamente, è possibile ripristinare i tipi Java di base invece degli oggetti e identificare il tipo di restituzione per il proprio utente.

Tenere presente che questo esempio deve inviare esplicitamente l'oggetto Java corretto.

```
  public class MessageEntryRecord extends Record
{
      static private RecordFormat format = new MessageEntryFormat();

      public MessageEntryRecord()
  {
          super(format);
  }

      public int getMessageNumber()
  {
          // Reperire il numero messaggio come int. Nota: si è a conoscenza del formato record e quindi
    // si conoscono i nomi dei campi. E' più sicuro richiamare il campo per nome nel caso in cui
    // sia stato inserito un campo nel formato a noi sconosciuto.
    return ((Integer)getField("msgnum")).intValue();
  }

      public String getMessageText()
  {
          // Restituire il testo del messaggio
    return (String)getField("msgtext");
  }

      public String getSender()
  {
          // Riportare il mittente del messaggio
    return (String)getField("sender");
  }

      public String getTimeSent()
  {
          // Riportare il mittente del messaggio
    return (String)getField("timesent");
  }

      // E' possibile aggiungere alcuni programmi di impostazione
}
```

Esempio: restituzione di un nuovo MessageEntryRecord

E' necessario sostituire il metodo getNewRecord() nella classe MessageEntryFormat (nell'esempio in alto) al fine di restituire un nuovo MessageEntryRecord.Per sostituire il metodo, aggiungere quanto segue alla classe MessageEntryFormat:

```
      public Record getNewRecord(byte[] data)
  {
          Record r = new MessageEntryRecord();
          r.setContents(data);
          return r;
  }
```

Dopo aver aggiunto il nuovo metodo getNewRecord(), è possibile utilizzare il MessageEntryRecord per interpretare la voce della coda dati:

```
    // Richiamare un oggetto record dal formato record, inizializzarlo con i dati che derivano dalla
    // voce appena letta.
Notare l'uso del nuovo metodo getNewRecord() sovrascritto.
    MessageEntryRecord rec = (MessageEntryRecord)entryFormat.getNewRecord(dqEntry.getData());

    // Emettere la voce completa come String. Il contenuto del record viene convertito in Oggetti Java
  // basati sul formato record della voce.
    System.out.println(rec.toString());
    // Richiamare il contenuto dei singoli campi della voce. Ogni contenuto del campo viene convertito
  // in un oggetto Java.
    int num = rec.getMessageNumber();    // Reperire il numero messaggio come int
    String s = rec.getSender();         // Reperire il mittente
    String text = rec.getMessageText(); // Reperire il testo del messaggio
    // Emettere i dati
  System.out.println(num + " " + s + " " + text);
```

Esempio: utilizzo delle classi DataQueue per inserire i dati in una coda

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per informazioni legali importanti.

```
///////////////////////////////////////////////////////////////////////////
//
// Esempio DataQueue. Questo programma utilizza la classe DataQueue per inserire un
// record in una coda dati.
//
// Questo esempio utilizza le classi Record e Record format per inserire dati
// nella coda. I dati di stringa vengono convertiti da Unicode in ebcdic
// e i numeri vengono convertiti da Java al formato server. Poiché i dati
// vengono convertiti le voci della coda dati possono essere lette da un programma server
// o da un programma iSeries Access per Windows ed anche da un altro programma Java.
//
// Ciò rappresenta il lato consumatore dell'esempio produttore/consumatore. Inserisce voci
// di lavoro nella coda perché il consumatore le elabori.
//
// Sintassi del comando:
//    DQProducerExample system
//
///////////////////////////////////////////////////////////////////////////


import java.io.*;
import java.util.*;
import java.net.*;
import com.ibm.as400.access.*;

public class DQProducerExample extends Object
{
      // Creare un programma di lettura per richiamare l'immissione dall'utente.
   static BufferedReader inputStream =
      new BufferedReader(new InputStreamReader(System.in),1);

     public static void main(String[] parameters)
   {
            System.out.println( " " );

            // se non è stato specificato il nome di sistema, visualizzare il testo di aiuto ed uscire.
      if (parameters.length >= 1)
      {
                                    try
         {
                     // Il primo parametro indica il sistema che contiene la coda dati.
                     String system = parameters[0];

                         // Creare un oggetto AS400 per il server che contiene la coda dati.
                        AS400 as400 = new AS400(system);
```

```
// Creare un formato record per il formato della voce coda dati.
// Questo formato corrisponde al formato nella classe DQConsumer. Un
// record consiste di:
//    - un numero a quattro byte -- il numero del cliente
//    - un numero a quattro byte -- il numero parte
//    - una stringa a 20 caratteri -- la descrizione parte
//    - un numero a quattro byte -- il numero delle parti in questo ordine
// Per prima cosa creare i tipi dati base.
BinaryFieldDescription customerNumber =
   new BinaryFieldDescription(new AS400Bin4(), "CUSTOMER_NUMBER");

            BinaryFieldDescription partNumber =
   new BinaryFieldDescription(new AS400Bin4(), "PART_NUMBER");

            CharacterFieldDescription partName =
   new CharacterFieldDescription(new AS400Text(20, as400), "PART_NAME");

            BinaryFieldDescription quantity =
   new BinaryFieldDescription(new AS400Bin4(), "QUANTITY");

            // Creare un formato record e compilarlo con i tipi dati base.
            RecordFormat dataFormat = new RecordFormat();
            dataFormat.addFieldDescription(customerNumber);
            dataFormat.addFieldDescription(partNumber);
            dataFormat.addFieldDescription(partName);
            dataFormat.addFieldDescription(quantity);

// Creare la libreria che contiene la coda dati
// utilizzando CommandCall.
            CommandCall crtlib = new CommandCall(as400);
crtlib.run("CRTLIB JAVADEMO");

            // Creare l'oggetto coda dati.
DataQueue dq = new DataQueue(as400, "/QSYS.LIB/JAVADEMO.LIB/PRODCONS.DTAQ");

            // Creare la coda dati solo nel caso che questa sia la prima volta
// che viene eseguito il programma. La coda esiste già, l'eccezione viene rilevata e
            // ignorata.
                        try
{
                  dq.create(96);
}
      catch (Exception e) {};

            // Richiamare il primo campo di dati dall'utente.
System.out.print("Enter customer number (or 0 to quit): ");
            int customer = getInt();

            // Mentre vi sono dati da inserire nella coda.
while (customer > 0)
{
                  // Richiamare il resto dei dati per questo ordine dall'utente.
   System.out.print("Enter part number: ");
                  int part = getInt();

   System.out.print("Enter quantity: ");
                  int quantityToOrder = getInt();

   String description = "part " + part;

   // Creare un record in base al formato record. Il record
                  // è vuoto al momento, alla fine conterrà i dati.
                  Record data = new Record(dataFormat);

                  // Impostare i valori ricevuti dall'utente nel record.
   data.setField("CUSTOMER_NUMBER", new Integer(customer));
```

```
            data.setField("PART_NUMBER",     new Integer(part));
            data.setField("QUANTITY",        new Integer(quantityToOrder));
            data.setField("PART_NAME", description);

            // Convertire il record in una schiera di byte. La schiera di byte
            // rappresenta ciò che è stato inserito nella coda dati.
                            byte [] byteData = data.getContents();

                     System.out.println("");
            System.out.println("Writing record to the server ...");
                     System.out.println("");

                            // Scrivere il record nella coda dati.
                            dq.write(byteData);

                            // Richiamare il successivo valore dall'utente.
            System.out.print("Enter customer number (or 0 to quit): ");
                            customer = getInt();
         }
      }
                                 catch (Exception e)
      {
         // Se una qualsiasi delle operazioni riportate sopra ha dato esito negativo
         // considerare non riuscita l'operazione della coda dati ed emettere l'eccezione.

         System.out.println("Data Queue operation failed");
                 System.out.println(e);
      }
   }

         // Visualizzare il testo di aiuto quando i parametri non sono corretti.
                           else
   {
                System.out.println("");
                System.out.println("");
                System.out.println("");
      System.out.println("Parameters are not correct. Command syntax is:");
                System.out.println("");
      System.out.println(" DQProducter system");
                System.out.println("");
                System.out.println("Where");
                System.out.println("");
      System.out.println(" system = Server that has the data queue");
                System.out.println("");
                System.out.println("For example:");
                System.out.println("");
      System.out.println(" DQProducerExample mySystem");
                System.out.println("");
                System.out.println("");
   }

           System.exit(0);
}

   // Questa è la sottoroutine che richiama una stringa di caratteri dall'utente
   // e la converte in un int.
   static int getInt()
{
        int i = 0;
                boolean Continue = true;

         while (Continue)
   {
                                 try
      {
                     String s = inputStream.readLine();
```

```
                        i = (new Integer(s)).intValue();
                              Continue = false;
         }
                                    catch (Exception e)
         {
                   System.out.println(e);
            System.out.print("Please enter a number ==>");
         }
      }

            return i;
   }
}
```

Esempio: utilizzo delle classi DataQueue per leggere le voci da una coda dati

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per informazioni legali importanti.

```
////////////////////////////////////////////////////////////////////////////
//
// Esempio DataQueue. Questo programma utilizza le classi DataQueue per leggere
// le voci da una coda dati sul server. Le voci sono state inserite sulla
// coda tramite il programma di esempio DQProducer.
//
// Ciò rappresenta il lato consumatore dell'esempio produttore/consumatore. Legge
// le voci dalla coda e le elabora.
//
// Sintassi del comando:
//    DQConsumerExample system
//
////////////////////////////////////////////////////////////////////////////


import java.io.*;
import java.util.*;
import java.net.*;
import com.ibm.as400.access.*;

public class DQConsumerExample extends Object
{
   public static void main(String[] parameters)
   {
      System.out.println( " " );

            // se non è stato specificato un nome di sistema, visualizzare il testo di aiuto ed uscire.
      if (parameters.length >= 1)
      {
                                    try
         {

                        // Il primo parametro indica il sistema che contiene la coda dati.
                        String system = parameters[0];

                            // Creare un oggetto AS400 per il server che contiene la coda dati.
                           AS400 as400 = new AS400(system);


             // Creare un formato record per il formato della voce coda dati.
             // Questo formato corrisponde al formato nella classe DQProducer. Un
             // record consiste di:
             //    - un numero a quattro byte -- il numero del cliente
             //    - un numero a quattro byte -- il numero parte
             //    - una stringa a 20 caratteri -- la descrizione parte
             //    - un numero a quattro byte -- il numero delle parti in questo ordine

             // Per prima cosa creare i tipi dati base.
             BinaryFieldDescription customerNumber =
```

```
          new BinaryFieldDescription(new AS400Bin4(), "CUSTOMER_NUMBER");

                    BinaryFieldDescription partNumber =
          new BinaryFieldDescription(new AS400Bin4(), "PART_NUMBER");

                    CharacterFieldDescription partName =
          new CharacterFieldDescription(new AS400Text(20, as400), "PART_NAME"

                    BinaryFieldDescription quantity =
          new BinaryFieldDescription(new AS400Bin4(), "QUANTITY"

                    // Creare un formato record e compilarlo con i tipi dati base.
                    RecordFormat dataFormat = new RecordFormat();

                    dataFormat.addFieldDescription(customerNumber);
                    dataFormat.addFieldDescription(partNumber);
                    dataFormat.addFieldDescription(partName);
                    dataFormat.addFieldDescription(quantity);

                    // Creare l'oggetto coda dati che rappresenta la coda dati sul
                              // sul server.
          DataQueue dq = new DataQueue(as400, "/QSYS.LIB/JAVADEMO.LIB/PRODCONS.DTAQ");

                   boolean Continue = true;

          // Leggere la prima voce dalla coda. Il valore del supero tempo è
                    // impostato su -1 quindi questo programma attenderà una voce a tempo indeterminato.
          System.out.println("*** Waiting for an entry for process ***");

                    DataQueueEntry DQData = dq.read(-1);

               while (Continue)
          {

             // Si legge semplicemente una voce dalla coda. Inserire i dati in
                             // un oggetto record in modo che il programma possa accedere ai campi dei
             // dati tramite nome. L'oggetto Record convertirà inoltre
                             // i dati dal formato server al formato Java.
                             Record data = dataFormat.getNewRecord(DQData.getData());

                             // Richiamare due valori dal record e visualizzarli.
             Integer amountOrdered = (Integer) data.getField("QUANTITY");
             String  partOrdered   = (String)  data.getField("PART_NAME");

             System.out.println("Need " + amountOrdered + " of "
                                + partOrdered);
                      System.out.println("");
             System.out.println("*** Waiting for an entry for process ***");

                             // Attendere la voce successiva.
                             DQData = dq.read(-1);
          }
       }
                                  catch (Exception e)
       {
          // Se una qualsiasi delle operazioni riportate sopra ha dato esito negativo
          // considerare non riuscita l'operazione della coda dati ed emettere l'eccezione.
          System.out.println("Data Queue operation failed");
                  System.out.println(e);
       }
    }

         // Visualizzare il testo di aiuto quando i parametri non sono corretti.
                           else
    {
                 System.out.println("");
                 System.out.println("");
```

```
                System.out.println("");
          System.out.println("Parameters are not correct. Command syntax is:");
                System.out.println("");
          System.out.println(" DQConsumerExample system");
                System.out.println("");
          System.out.println("Where");
                System.out.println("");
          System.out.println(" system = Server that has the data queue");
                System.out.println("");
          System.out.println("For example:");
                System.out.println("");
          System.out.println(" DQConsumerExample mySystem");
                System.out.println("");
                System.out.println("");
       }

       System.exit(0);
    }
}
```

Utilizzo dell'esempio dei tipi di dati

E' possibile utilizzare le classi AS400DataType con ProgramCall per fornire i dati ai parametri del programma e per interpretare i dati restituiti nei parametri del programma.

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per informazioni legali importanti.

Esempio: utilizzo delle classi AS400DataType con ProgramCall

Il seguente esempio mostra come utilizzare le classi AS400DataType tramite l'utilizzo di ProgramCall per richiamare l'API di sistema, QUSRMBRD "Richiamare descrizione membro". L'API QUSRMBRD richiama la descrizione di un membro specifico in un file di database. Le tabelle che seguono l'esempio elencano i parametri QUSRMBRD necessari e le informazioni richiamate dall'esempio.

```
// Creare un oggetto ProgramCall. Verrà impostato successivamente il nome del programma
// e l'elenco parametri.
  ProgramCall qusrmbrd = new ProgramCall(new AS400());

  // Creare un elenco di parametri programma vuoto
  ProgramParameter[] parms = new ProgramParameter[6];

// Creare AS400DataTypes per convertire i parametri di immissione da tipi Java
// in dati server
  AS400Bin4 bin4 = new AS400Bin4();

// E' necessario un oggetto AS400Text separato per ogni parametro con una
// lunghezza differente poiché la classe AS400Text richiede che venga specificata la
// lunghezza.
  AS400Text char8Converter = new AS400Text(8)
  AS400Text char20Converter = new AS400Text(20);
  AS400Text char10Converter = new AS400Text(10);
  AS400Text char1Converter = new AS400Text(1);

// Popolare l'elenco di parametri; verranno utilizzati gli oggetti AS400DataType
// per convertire i valori Java in schiere di byte contenenti i dati server.

// Per i parametri di emissione è necessario solo specificare il numero di byte
// restituiti
parms[0] = new ProgramParameter(135);
  parms[1] = new ProgramParameter(bin4.toBytes(new Integer(135)));
parms[2] = new ProgramParameter(char8Converter.toBytes("MBRD0100"));
parms[3] = new ProgramParameter(char20Converter.toBytes("MYFILE    MYLIB     "));
parms[4] = new ProgramParameter(char10COnverter.toBytes("MYMEMBER  "));
parms[5] = new ProgramParameter(char1Converter.toBytes("0"));
```

```
      // Impostare il nome di programma e l'elenco di parametri
    qusrmbrd.setProgram("/qsys.lib/qusrmbrd.pgm", parms);

      // Chiamare il programma
                               try
    {
         qusrmbrd.run();
    }
      catch(Exception e)
    {
         // Gestire qualsiasi eccezione
    }

    // Richiamare le informazioni recuperate. Notare che si tratta di dati server grezzi.
      byte[] receiverVar = parms[0].getOutputData();


  // E' necessario per convertire i dati ora e data
    AS400Text char13Converter = new AS400Text(13);

  // E' necessario per convertire i dati di descrizione testo
    AS400Text char50Converter = new AS400Text(50);

      // Creare un AS400Structure per gestire le informazioni restituite
      AS400DataType[] dataTypeArray = new AS400DataType[11];
      dataTypeArray[0] = bin4;
      dataTypeArray[1] = bin4;
      dataTypeArray[2] = char10Converter;
      dataTypeArray[3] = char10Converter;
      dataTypeArray[4] = char10Converter;
      dataTypeArray[5] = char10Converter;
      dataTypeArray[6] = char10Converter;
      dataTypeArray[7] = char13Converter;
      dataTypeArray[8] = char13Converter;
      dataTypeArray[9] = char50Converter;
      dataTypeArray[10] = char1Converter;
      AS400Structure returnedDataConverter = new AS400Structure(dataTypeArray);


      // Convertire i dati restituiti in una schiera di Oggetti Java utilizzando
      // returnedDataConverter
      Object[] qusrmbrdInfo = dataConverter.toObject(receiverVar, 0);


      // Richiamare il numero di byte restituiti
      Integer bytesReturned = (Integer)qusrmbrdInfo[0];
      Integer bytesAvailable = (Integer)qusrmbrdInfo[1];
      if (bytesReturned.intValue() != 135)
    {
      System.out.println("Wrong amount of information returned.");
              System.exit(0);
    }
      String fileName = (String)qusrmbrdInfo[2];
      String libName = (String)qusrmbrdInfo[3];
      String mbrName = (String)qusrmbrdInfo[4];
      String fileAttribute = (String)qusrmbrdInfo[5];
      String sourceType = (String)qusrmbrdInfo[6];
      String created = (String)qusrmbrdInfo[7];
      String lastChanged = (String)qusrmbrdInfo[8];
      String textDesc = (String)qusrmbrdInfo[9];
      String isSourceFile = (String)qusrmbrdInfo[10];

      // Si emetteranno semplicemente tutte le informazioni sullo schermo
    System.out.println(fileName + " " + libName + " " + mbrName + " " +
                       fileAttribute + sourceType + " " + created + " " +
                       lastChanged + " " + textDesc + " " + isSourceFile);
```

La seguente tabella elenca i parametri necessari per l'API QUSRMBRD come quelli utilizzati nel precedente esempio.

Parametro QUSRMBRD	Immissione o emissione	Tipo	Descrizione
Variabile ricevente	Emissione	Char(*)	Buffer di carattere che contiene le informazioni richiamate.
Lunghezza della variabile ricevente	Immissione	Bin(4)	Lunghezza del buffer di carattere fornita per la variabile del ricevente.
Nome formato	Immissione	Char(8)	Formato che specifica le informazioni da richiamare. Deve essere uno dei seguenti: • MBRD0100 • MBRD0200 • MBRD0300 L'esempio che segue specifica MBRD0100.
Nome file database completo	Immissione	Char(20)	Il nome file completo. Questo è il nome file costituito da 10 spazi vuoti seguito dal nome libreria con 10 spazi vuoti. Per il nome libreria sono consentiti i valori speciali di *CURLIB e *LIBL.
Nome membro database	Immissione	Char(10)	Il nome del membro con 10 spazi vuoti. Sono consentiti i valori speciali *FIRST e *LAST.
Elaborazione di sostituzione	Immissione	Char(1)	Specifica se le sostituzioni devono essere elaborate. 0 indica che le sostituzioni non devono essere elaborate. Questo è il valore da specificare.

La tabella che segue elenca le informazioni richiamate dall'esempio (in base al formato MBRD0100, secondo quanto specificato nel precedente esempio):

Informazioni richiamate	Tipo
Byte restituiti	Bin(4)
Byte disponibili	Bin(4)
Nome file database	Char(10)
Nome libreria file database	Char(10)
Nome membro	Char(10)
Attributo file (tipo di file: PF, LF, DDMF)	Char(10)
Tipo sorgente (tipo del membro sorgente se questo è un file sorgente)	Char(10)
Data e ora di creazione	Char(13)

Informazioni richiamate	Tipo
Data e ora dell'ultima modifica dell'origine	Char(13)
Descrizione del testo membro	Char(50)
File sorgente (se il file è un file sorgente: 0=file di dati, 1=file sorgente)	Char(1)

Esempio: utilizzo di KeyedDataQueue

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per informazioni legali importanti.

```
////////////////////////////////////////////////////////////////////////////
//
// Esempio DataQueue. Questo programma utilizza la classe KeyedDataQueue per inserire un
// record in una coda dati.
//
// La chiave è un numero e i dati sono rappresentati da una stringa Unicode. Questo programma
// presenta un modo per convertire un int in una schiera di byte e per convertire
// una stringa Java in una schiera di byte in modo che sia possibile scriverla nella coda.
//
// Ciò rappresenta il lato consumatore dell'esempio produttore/consumatore. Inserisce voci di
// di lavoro nella coda perché il consumatore le elabori.
//
// Sintassi del comando:
// DQKeyedProducer system
//
////////////////////////////////////////////////////////////////////////////

import java.io.*;
import java.util.*;
import java.net.*;
import com.ibm.as400.access.*;

public class DQKeyedProducer extends Object
{

      // Creare un programma di lettura per richiamare l'immissione dall'utente.

   static BufferedReader inputStream =
      new BufferedReader(new InputStreamReader(System.in),1);



     public static void main(String[] parameters)
   {

      System.out.println( " " );



            // se non è stato specificato il nome di sistema, visualizzare il testo di aiuto ed uscire.

      if (parameters.length >= 1)
      {

                  // Il primo parametro indica il sistema che contiene la coda dati.

                  String system = parameters[0];



         System.out.println("Priority is a numeric value. The value ranges are:");
         System.out.println("  0 -  49 = low priority");
         System.out.println(" 50 - 100 = medium priority");
         System.out.println("100 +     = high priority");
```

```
          System.out.println( " " );

                                try
        {
                        // Creare un oggetto AS400 per il server che contiene la coda dati.

                       AS400 as400 = new AS400(system);



            // Utilizzare CommandCall per creare la libreria che contiene la
            // coda dati.

                        CommandCall crtlib = new CommandCall(as400);
            crtlib.run("CRTLIB JAVADEMO");



                        // Creare l'oggetto coda dati.

            QSYSObjectPathName name = new QSYSObjectPathName("JAVADEMO", "PRODCON2", "DTAQ");

                        KeyedDataQueue dq = new KeyedDataQueue(as400, name.getPath());



                        // Creare la coda dati solo nel caso che questa sia la prima volta
            // che viene eseguito il programma. La coda esiste già, l'eccezione viene rilevata e
                        // ignorata.
La lunghezza della chiave è di quattro byte, la lunghezza
                        // di una voce è di 96 byte.

                                    try
            {
                              dq.create(4, 96);
            }
                  catch (Exception e) {};



                        // Richiamare i dati dall'utente.

            System.out.print("Enter message: ");
                        String message = inputStream.readLine();

            System.out.print("Enter priority: ");
                        int priority = getInt();



                        // Mentre vi sono dati da inserire nella coda.

            while (priority > 0)
            {

               // Si desidera scrivere una stringa java come voce nella coda.
               // Tuttavia, l'immissione nella coda dati è una schiera di byte, quindi convertire
               // la stringa in una schiera di byte.

               byte [] byteData = message.getBytes("UnicodeBigUnmarked");



               // La chiave è un numero. L'immissione nella coda di dati è una schiera
                               // di byte, quindi convertire l'int in una schiera di byte;
```

```
                        byte [] byteKey = new byte[4];
            byteKey[0] = (byte) (priority >>> 24);
            byteKey[1] = (byte) (priority >>> 16);
            byteKey[2] = (byte) (priority >>> 8);
                        byteKey[3] = (byte) (priority);


                    System.out.println("");
            System.out.println("Writing record to the server...");
                    System.out.println("");



                        // Scrivere il record nella coda dati.

                        dq.write(byteKey, byteData);



                        // Richiamare il successivo valore dall'utente.

            System.out.print("Enter message: ");
                        message = inputStream.readLine();

            System.out.print("Enter priority: ");
                        priority = getInt();
         }
      }
                               catch (Exception e)
      {

         // Se una qualsiasi delle operazioni riportate sopra ha dato esito negativo
         // considerare non riuscita l'operazione ed emettere l'eccezione.

         System.out.println("Data Queue operation failed");
                 System.out.println(e);
      }
   }


         // Visualizzare il testo di aiuto quando i parametri non sono corretti.

                          else
   {
               System.out.println("");
               System.out.println("");
               System.out.println("");
      System.out.println("Parameters are not correct.  Command syntax is:");
               System.out.println("");
      System.out.println("  DQKeyedProducter system");
               System.out.println("");
               System.out.println("Where");
               System.out.println("");
      System.out.println("  system = server that has the data queue");
               System.out.println("");
               System.out.println("For example:");
               System.out.println("");
      System.out.println("  DQKeyedProducer mySystem");
               System.out.println("");
               System.out.println("");
   }

            System.exit(0);
}
```

```
      // Questa è la sottoroutine che richiama una stringa di caratteri dall'utente
      // e la converte in un int.

      static int getInt()
   {
          int i = 0;
                  boolean Continue = true;

            while (Continue)
      {

                                      try
         {
                        String s = inputStream.readLine();

                        i = (new Integer(s)).intValue();
                              Continue = false;
         }
                                      catch (Exception e)
         {
                   System.out.println(e);
            System.out.print("Please enter a number ==>");
         }
      }

            return i;
   }
}
```

Esempio: utilizzo delle classi KeyedDataQueue per leggere le voci da una coda dati.

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per informazioni legali importanti.

```
////////////////////////////////////////////////////////////////////////////
//
// Esempio Keyed Data Queue. Questo programma utilizza le classi KeyedDataQueue per
// leggere le voci dalla coda dati sul server. Le voci sono state inserite sulla
// coda con il programma di esempio DQKeyedProducer.
//
// La chiave è un numero e i dati sono rappresentati da una stringa unicode. Questo programma
// mostra un modo per convertire la schiera di byte in un int Java e per leggere
// un schiera di byte e convertirla in una stringa Java.
//
// Ciò rappresenta il lato consumatore dell'esempio produttore/consumatore. Legge
// le voci dalla coda e le elabora.
//
// Sintassi del comando:
//    DQKeyedConsumer system
//
////////////////////////////////////////////////////////////////////////////

import java.io.*;
import java.util.*;
import java.net.*;
import com.ibm.as400.access.*;

public class DQKeyedConsumer extends Object
{
   public static void main(String[] parameters)
   {
      System.out.println( " " );

            // se non è stato specificato un nome di sistema, visualizzare il testo di aiuto ed uscire.
      if (parameters.length >= 1)
      {
```

```
         // Il primo parametro indica il sistema che contiene la coda dati.
         String system = parameters[0];

         // Creare schiere di byte per i limiti di priorità:
         // 100 +    = priorità alta
         // 50 - 100 = priorità media
         //  0 -  49 = priorità bassa

         byte [] key0 = new byte[4];
         key0[0] = 0;
         key0[1] = 0;
         key0[2] = 0;
         key0[3] = 0;

         byte [] key50  = new byte[4];
key50[0] = (byte) (50 >>> 24);
key50[1] = (byte) (50 >>> 16);
key50[2] = (byte) (50 >>> 8);
         key50[3] = (byte) (50);

         byte [] key100 = new byte[4];
key100[0] = (byte) (100 >>> 24);
key100[1] = (byte) (100 >>> 16);
key100[2] = (byte) (100 >>> 8);
         key100[3] = (byte) (100);

                           try
{
                 // Creare un oggetto AS400 per il server che contiene la coda dati.
                AS400 as400 = new AS400(system);

                 // Creare l'oggetto coda dati che rappresenti la coda dati
                 // sul server.

    QSYSObjectPathName name = new QSYSObjectPathName("JAVADEMO",
                                                     "PRODCON2",
                                                     "DTAQ");
                 KeyedDataQueue dq = new KeyedDataQueue(as400, name.getPath());
                 KeyedDataQueueEntry DQData = null;

                                 try
    {
                   boolean Continue = true;

                       // Continuare fino alla chiusura da parte dell'utente.
            while (Continue)
       {
          // Ricercare una voce a priorità alta nella coda. Se ne viene individuata una,
          // elaborare tale voce. Notare che il metodo peek non elimina
          // la voce se ne viene individuata una. Inoltre, notare che il superotempo è pari a
          // 0. Se non viene rilevata alcuna voce il controllo restituisce una voce
          // coda dati nulla.
          DQData = dq.read(key100, 0, "GE");

                                  if (DQData != null)
          {
                                        processEntry(DQData);
          }

          // non è stata individuata alcuna voce a priorità alta. Ricercare una voce
                           // a priorità media.
                                else
          {
             DQData = dq.read(key50, 0, "GE");

                                     if (DQData != null)
```

```
                {
                                                processEntry(DQData);
                }

                // in caso non siano state individuate voci a priorità media, ricercare una voce
                // a priorità media.
                                          else
                {
                   DQData = dq.read(key0, 0, "GE");

                                             if (DQData != null)
                   {
                                                   processEntry(DQData);
                   }

                                            else
                   {
                     System.out.println("Nothing to process, will check again in 30 seconds");
                                                   Thread.sleep(30000);
                   }
                }
             }
          }
       }
                                      catch (Exception e)
       {
          // Se una qualsiasi delle operazioni riportate sopra ha dato esito negativo
          // considerare non riuscita l'operazione della coda dati ed emettere l'eccezione.
          System.out.println("Could not read from the data queue.");
                  System.out.println(e);
       };

    }
                                 catch (Exception e)
    {
        // Se una qualsiasi delle operazioni riportate sopra ha dato esito negativo
        // considerare non riuscita l'operazione della coda dati ed emettere l'eccezione.

        System.out.println("Data Queue operation failed");
                System.out.println(e);
    }
  }

       // Visualizzare il testo di aiuto quando i parametri non sono corretti.
                        else
  {
             System.out.println("");
             System.out.println("");
             System.out.println("");
     System.out.println("Parameters are not correct. Command syntax is:");
             System.out.println("");
     System.out.println(" DQKeyedConsumer system");
             System.out.println("");
             System.out.println("Where");
             System.out.println("");
     System.out.println(" system = Server that has the data queue");
             System.out.println("");
             System.out.println("For example:");
             System.out.println("");
             System.out.println("");
     System.out.println(" DQKeyedConsumer mySystem");
             System.out.println("");
             System.out.println("");
  }

         System.exit(0);
}
```

```
      static void processEntry(KeyedDataQueueEntry DQData)
   {
                                try
      {
          // I dati sono rappresentati da una stringa. Estrarre la stringa dalla voce della coda dati.
                    // Nella voce della coda dati i dati costituiscono una schiera di byte quindi convertire la
                    // voce da schiera di byte in stringa.
          String message  = new String(DQData.getData(), "UnicodeBig");

          // La chiave è una schiera di byte. Estrarre la chiave dalla voce della coda dati
                    // e convertirla in un numero.
                    byte [] keyData = DQData.getKey();

                    int keyValue = ((keyData[0] & 0xFF) << 24) +
                                                ((keyData[1] & 0xFF) << 16) +
                                                ((keyData[2] & 0xFF) <<  8) +
                                                  (keyData[3] & 0xFF);

                    // Emettere la voce.
          System.out.println("Priority: " + keyValue + "   message: " + message);

      }
                                catch (Exception e)
      {
         // Se una qualsiasi delle operazioni riportate sopra ha dato esito negativo considerare
         // non riuscita l'operazione della coda dati ed emettere l'eccezione.

         System.out.println("Could not read from the data queue");
                 System.out.println(e);
      }
   }
}
```

Esempi: utilizzo di IFSFile

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per informazioni legali importanti.

Gli esempi che seguono mostrano come utilizzare la classe IFSFile:

- Esempio: creazione di un indirizzario
- Esempio: utilizzo delle eccezioni IFSFile per la traccia degli errori
- Esempio: elenco dei file con estensione .txt
- "Esempio: utilizzo del metodo IFSFile listFiles() per elencare il contenuto di un indirizzario" a pagina 496

Esempio: creazione di un indirizzario

```
                           // Creare un oggetto AS400.
Questo nuovo
                                          // indirizzario verrà creato su
                                          // iSeries.
     AS400 sys = new AS400("mySystem.myCompany.com");

                                          // Creare un oggetto file che
                                          // rappresenta l'indirizzario.
          IFSFile aDirectory = new IFSFile(sys, "/mydir1/mydir2/newdir");

                                          // Creare l'indirizzario.
           (aDirectory.mkdir())
                System.out.println("Create directory was successful");

                                          // Altrimenti la creazione dell'indirizzario ha avuto esito negativo
                           else
     {
```

```
                                        // Se l'oggetto esiste già,
                                        // scoprire se è un indirizzario o
                                        // un file, quindi visualizzare un messaggio.
            if (aDirectory.exists())
       {
                  if (aDirectory.isDirectory())
                        System.out.println("Directory already exists");
                              else
                        System.out.println("File with this name already exists");
       }
                           else
                  System.out.println("Create directory failed");
     }

                                        // Scollegarsi in quanto è terminata
                                        // file è terminato.
         sys.disconnectService(AS400.FILE);
```

Esempio: utilizzo delle eccezioni IFSFile per tenere traccia degli errori

Quando si verifica un errore, la classe IFSFile invia l'eccezione ExtendedIOException.Questa eccezione contiene un codice di ritorno che indica la causa dell'errore. La classe IFSFile invia l'eccezione anche quando la classe java.io duplicata da IFSFile non lo fa. Ad esempio, il metodo di cancellazione da java.io.File restituisce un valore booleano per indicare l'esito positivo o negativo. Il metodo di cancellazione in IFSFile restituisce un valore booleano, ma se la cancellazione non riesce, viene lanciata una ExtendedIOException. ExtendedIOException fornisce al programma Java informazioni dettagliate relative al motivo per cui la cancellazione ha avuto esito negativo.

```
                          // Creare un oggetto AS400.
     AS400 sys = new AS400("mySystem.myCompany.com");

                                        // Creare un oggetto file che
                                        // rappresenta il file.
          IFSFile aFile = new IFSFile(sys, "/mydir1/mydir2/myfile");

                       // Cancellare il file.
                              try
     {
               aFile.delete();

                                        // La cancellazione ha avuto esito positivo.
               System.out.println("Delete successful ");
     }

                                        // Cancellazione non riuscita.
Richiamare il codice di
                                        // di ritorno dall'eccezione e
                                        // visualizzare il motivo della mancata cancellazione.
          catch (ExtendedIOException e)
     {
               int rc = e.getReturnCode();

               switch (rc)
       {
                     case ExtendedIOException.FILE_IN_USE:
                           System.out.println("Delete failed, file is in use ");
                                 break;

                     case ExtendedIOException.PATH_NOT_FOUND:
                           System.out.println("Delete failed, path not found ");
                                 break;

                                             // ... per ogni errore specifico di cui
                                             // si desidera tenere traccia.

                  default:
```

```
                        System.out.println("Delete failed, rc = ");
                        System.out.println(rc);
      }
   }
```

Esempio: elenco dei file con un'estensione .txt

Il programma Java può specificare facoltativamente criteri di corrispondenza quando elenca i file nell'indirizzario. I criteri di corrispondenza riducono il numero di file restituiti dal server all'oggetto IFSFile, che migliora le prestazioni. L'esempio che segue mostra come elencare file con l'estensione .txt:

```
                                        // Creare l'oggetto AS400.
        AS400 system = new AS400("mySystem.myCompany.com");

                                        // Creare l'oggetto file.
        IFSFile directory = new IFSFile(system, "/");

                                        // Creare un elenco di tutti i file con
                                        // estensione .txt
        String[] names = directory.list("*.txt");

                                        // Visualizzare i nomi.
        if (names != null)
            for (int i = 0; i < names.length; i++)
                System.out.println(names[i]);
                        else
            System.out.println("No .txt files");
```

Esempio: utilizzo del metodo IFSFile listFiles() per elencare il contenuto di un indirizzario

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per informazioni legali importanti.

```
////////////////////////////////////////////////////////////////////////////////
//
// Esempio IFSListFiles. Questo programma utilizza le classi IFS
// per elencare il contenuto di un indirizzario nel server.
//
// Sintassi del comando:
//    IFSListFiles system directory
//
// Ad esempio,
//    IFSListFiles MySystem /path1
//
////////////////////////////////////////////////////////////////////////////////

import java.io.*;
import java.util.*;
import com.ibm.as400.access.*;

public class IFSListFiles extends Object
{
      public static void main(String[] parameters)
    {
              System.out.println( " " );

              String directoryName = "";
              String system        = "";


              // se entrambi i parametri non sono stati specificati, visualizzare il testo di aiuto ed uscire.

        if (parameters.length >= 2)
        {

                    // Supponiamo che il primo parametro sia il nome di sistema ed
```

```
                    // il secondo parametro sia il nome di indirizzario

                       system = parameters[0];
                    directoryName = parameters[1];

                                    try
            {
                            // Creare un oggetto AS400 per il server che contiene i file.

                            AS400 as400 = new AS400(system);



                            // Creare l'oggetto IFSFile per l'indirizzario.

                            IFSFile directory = new IFSFile(as400, directoryName);


                // Creare un elenco di IFSFiles. Inoltrare al metodo listFiles
                // l'oggetto filtro indirizzario ed i criteri di
                //  ricerca. Tale metodo memorizza in cache le informazioni sull'attributo. Ad
                // esempio, quando viene chiamato isDirectory() su un oggetto IFSFile
                // nella schiera di file restituita nel seguente codice,
                // non è necessaria alcuna chiamata al server.
                //
                // Tuttavia, con l'utente del metodo listFiles, le informazioni
                // sull'attributo non verranno aggiornate automaticamente dal
                // server.
Ciò significa che le informazioni sull'attributo potrebbero non essere
                // coerenti con le informazioni sul server.

                                IFSFile[] directoryFiles = directory.listFiles(new MyDirectoryFilter(),"*");

                // Informare l'utente se l'indirizzario non esiste o risulta vuoto

                                if (directoryFiles == null)
                {
                                   System.out.println("The directory does not exist");
                    return;
                }

                                else if (directoryFiles.length == 0)
                {
                                   System.out.println("The directory is empty");
                    return;
                }

                                for (int i=0; i< directoryFiles.length; i++)
                    {
                                            // Stampare le informazioni nell'elenco.
                                           // Stampare il nome del file corrente

                                            System.out.print(directoryFiles[i].getName());


                                 // Ingrandire l'emissione in modo che le colonne si allineino

                            for (int j = directoryFiles[i].getName().length(); j <18; j++)
                                        System.out.print("   ");


                                 // Stampare la data in cui il file è stato modificato l'ultima volta.

                                            long changeDate = directoryFiles[i].lastModified();
                                 Date d = new Date(changeDate);
                                 System.out.print(d);
                                 System.out.print("   ");
```

```
                                // Stampare se la voce è un file o un indirizzario

                                System.out.print("   ");

                                            if (directoryFiles[i].isDirectory())
                                    System.out.println("");
                                            else
                                                    System.out.println(directoryFiles[i].length());

                        }
                    }

                                       catch (Exception e)
                    {
                        // Se una qualsiasi delle operazioni riportate sopra ha dato esito negativo
                        // considerare non riuscito l'elenco ed emettere l'eccezione.

                                    System.out.println("List failed");
                        System.out.println(e);
                    }
                }

                // Visualizzare il testo di aiuto quando i parametri non sono corretti.

                                    else
                {
                             System.out.println("");
                             System.out.println("");
                             System.out.println("");
                    System.out.println("Parameters are not correct.  Command syntax is:");
                             System.out.println("");
                                System.out.println("   IFSListFiles as400 directory");
                             System.out.println("");
                             System.out.println("Where");
                             System.out.println("");
                             System.out.println("   as400 = system that contains the files");
                             System.out.println("   directory = directory to be listed");
                             System.out.println("");
                             System.out.println("For example:");
                             System.out.println("");
                                System.out.println("   IFSListFiles mySystem /dir1/dir2");
                             System.out.println("");
                             System.out.println("");
                }

                System.exit(0);
            }
        }


        ////////////////////////////////////////////////////////////////////////
        //
        // La classe filtro indirizzario stampa informazioni dall'oggetto file.
        //
        // Un altro metodo per utilizzare il filtro è quello di restituire true o false
        // a seconda delle informazioni contenute nell'oggetto file. Ciò consente alla funzione
        // mainline di decidere cosa fare riguardo all'elenco di file che corrispondono ai
        // criteri di ricerca.
        //
        ////////////////////////////////////////////////////////////////////////



        class MyDirectoryFilter implements IFSFileFilter
        {
```

```
       public boolean accept(IFSFile file)
    {
                                    try
        {
            // Conservare questa voce. La restituzione di true indica all'oggetto IFSList
                     // di restituire questo file nell'elenco di voci restituito nel
                     // metodo .list().

                     return true;
        }

                                    catch (Exception e)
        {
                     return false;
        }
    }
}
```

Esempio: utilizzo delle classi IFS per copiare un file da un indirizzario ad un altro.

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per informazioni legali importanti.

```
////////////////////////////////////////////////////////////////////////////
//
// Esempio IFSCopyFile. Questo programma utilizza le classi file system installabili
// per copiare un file da un indirizzario ad un altro sul server.
//
// Sintassi del comando:
//    IFSCopyFile system sourceName TargetName
//
// Ad esempio,
//    IFSCopyFile MySystem  /path1/path2/file.ext  /path3/path4/path5/file.ext
//
////////////////////////////////////////////////////////////////////////////



import java.io.*;
import java.util.*;
import com.ibm.as400.access.*;

public class IFSCopyFile extends Object
{
     public static void main(String[] parameters)
   {
            System.out.println( " " );

      String sourceName = "";
      String targetName = "";
            String system        = "";
            byte[] buffer     = new byte[1024 * 64];

            IFSFileInputStream  source = null;
            IFSFileOutputStream target = null;

            // Se i tre parametri non sono stati specificati, visualizzare il testo di aiuto ed uscire.

      if (parameters.length > 2)
      {

                  // Supponiamo che il primo parametro sia il nome di sistema,
                  // il secondo parametro sia il nome origine ed
                  // il terzo parametro quello di destinazione.

                     system = parameters[0];
                  sourceName = parameters[1];
                  targetName = parameters[2];
```

```
                            try
{
                // Creare un oggetto AS400 per il server che contiene i file.

                AS400 as400 = new AS400(system);



                // Aprire il file sorgente per l'accesso esclusivo.

   source = new IFSFileInputStream(as400, sourceName, IFSFileInputStream.SHARE_NONE);

   System.out.println("Source file successfully opened");



                // Aprire il file di destinazione per l'accesso esclusivo.

   target = new IFSFileOutputStream(as400, targetName, IFSFileOutputStream.SHARE_NONE, false);

   System.out.println("Target file successfully opened");



                // Leggere i primi 64K byte dal file sorgente.

                int bytesRead = source.read(buffer);



                // Poiché vi sono dati nel file sorgente copiare tali dati dal
                // file sorgente in quello di destinazione.

   while (bytesRead > 0)
   {
                      target.write(buffer, 0, bytesRead);
                      bytesRead = source.read(buffer);
   }

   System.out.println("Data successfully copied");



                // Effettuare la ripulitura chiudendo i file sorgente e destinazione.

                source.close();
                target.close();



                // Richiamare la data e l'ora dell'ultima modifica dal file sorgente e
                // impostarla su quello di destinazione.

                IFSFile src = new IFSFile(as400, sourceName);
                long dateTime = src.lastModified();

                IFSFile tgt = new IFSFile(as400, targetName);
                tgt.setLastModified(dateTime);

   System.out.println("Date/Time successfully set on target file");
   System.out.println("Copy Successful");

}
                            catch (Exception e)
{
   // Se una qualsiasi delle operazioni riportate sopra ha dato esito negativo
```

```
            // considerare non riuscita la copia ed emettere l'eccezione.

            System.out.println("Copy failed");
                    System.out.println(e);
         }
      }

      // Visualizzare il testo di aiuto quando i parametri non sono corretti.

                              else
      {
                  System.out.println("");
                  System.out.println("");
                  System.out.println("");
         System.out.println("Parameters are not correct.  Command syntax is:");
                  System.out.println("");
         System.out.println("   IFSCopyFile as400 source target");
                  System.out.println("");
                  System.out.println("Where");
                  System.out.println("");
                  System.out.println("   as400 = system that contains the files");
         System.out.println("   source = source file in /path/path/name format");
         System.out.println("   target = target file in /path/path/name format");
                  System.out.println("");
                  System.out.println("For example:");
                  System.out.println("");
         System.out.println("   IFSCopyFile myAS400  /dir1/dir2/a.txt  /dir3/b.txt");
                  System.out.println("");
                  System.out.println("");
      }

      System.exit(0);
   }
}
```

Esempio: utilizzo delle classi IFS per elencare il contenuto di un indirizzario

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per informazioni legali importanti.

```
///////////////////////////////////////////////////////////////////////////////
//
// Esempio IFSListFile. Questo programma utilizza le classi IFS
// per elencare il contenuto di un indirizzario sul server.
//
// Sintassi del comando:
//    IFSList system directory
//
// Ad esempio,
//    IFSList MySystem /path1
//
///////////////////////////////////////////////////////////////////////////////

import java.io.*;
import java.util.*;
import com.ibm.as400.access.*;

public class IFSList extends Object
{
     public static void main(String[] parameters)
   {
            System.out.println( " " );

            String directoryName = "";
            String system        = "";


            // se entrambi i parametri non sono stati specificati, visualizzare il testo di aiuto ed uscire.
```

```
      if (parameters.length >= 2)
      {

                  // Supponiamo che il primo parametro sia il nome di sistema ed
                  // il secondo parametro sia il nome di indirizzario

                     system = parameters[0];
                  directoryName = parameters[1];

                                    try
         {
                        // Creare un oggetto AS400 per il server che contiene i file.

                        AS400 as400 = new AS400(system);



                        // Creare l'oggetto IFSFile per l'indirizzario.

                        IFSFile directory = new IFSFile(as400, directoryName);



               // Creare l'elenco di nomi. Inoltrare al metodo elenco
                        // l'oggetto filtro indirizzario ed i criteri di ricerca corrispondenti.
               //
                        // Nota - questo esempio esegue l'elaborazione nell'oggetto
               // AS400TreePane.
in alternativa, è possibile elaborare l'elenco dopo
                        // che viene restituito dalla chiamata al metodo elenco.

                        String[] directoryNames = directory.list(new MyDirectoryFilter(),"*");



               // Informare l'utente se l'indirizzario non esiste o risulta vuoto

                        if (directoryNames == null)
                              System.out.println("The directory does not exist");

                        else if (directoryNames.length == 0)
                              System.out.println("The directory is empty");
         }

                                    catch (Exception e)
         {
               // Se una qualsiasi delle operazioni riportate sopra ha dato esito negativo
               //considerare non riuscito l'elenco ed emettere l'eccezione.

                        System.out.println("List failed");
               System.out.println(e);
         }
      }

      // Visualizzare il testo di aiuto quando i parametri non sono corretti.

                           else
      {
                  System.out.println("");
                  System.out.println("");
                  System.out.println("");
            System.out.println("Parameters are not correct.  Command syntax is:");
                  System.out.println("");
                  System.out.println("   IFSList as400 directory");
                  System.out.println("");
                  System.out.println("Where");
```

```
                System.out.println("");
                System.out.println("   as400 = system that contains the files");
                System.out.println("   directory = directory to be listed");
                System.out.println("");
                System.out.println("For example:");
                System.out.println("");
                System.out.println("   IFSCopyFile mySystem /dir1/dir2");
                System.out.println("");
                System.out.println("");
      }

      System.exit(0);
   }
}


////////////////////////////////////////////////////////////////////////
//
// La classe filtro indirizzario stampa informazioni dall'oggetto file.
//
// Un altro metodo per utilizzare il filtro è quello di restituire true o false
// a seconda delle informazioni contenute nell'oggetto file. Ciò consente alla funzione
// mainline di decidere cosa fare riguardo all'elenco di file che corrispondono ai
// criteri di ricerca.
//
////////////////////////////////////////////////////////////////////////



class MyDirectoryFilter implements IFSFileFilter
{
      public boolean accept(IFSFile file)
   {
                                 try
      {
                               // Stampare il nome del file corrente

                System.out.print(file.getName());



                // Ingrandire l'emissione in modo che le colonne si allineino

                for (int i = file.getName().length(); i < 18; i++)
                   System.out.print("   ");



                // Stampare la data in cui il file è stato modificato l'ultima volta.

                long changeDate = file.lastModified();
                Date d = new Date(changeDate);
                System.out.print(d);
                System.out.print("   ");



                // Stampare se la voce è un file o un indirizzario

                System.out.print("   ");

                if (file.isDirectory())
           System.out.println("<DIR>");
                            else
                System.out.println(file.length());
```

```
         // Conservare questa voce. La restituzione di true indica all'oggetto IFSList
                  // di restituire questo file nell'elenco di voci restituito nel
                  // metodo .list().

                  return true;
      }

                                  catch (Exception e)
      {
                  return false;
      }
   }
}
```

Esempio: utilizzo di JDBCPopulate per creare e popolare una tabella

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per informazioni legali importanti.

```
///////////////////////////////////////////////////////////////////////////////
//
// Esempio JDBCPopulate. Questo programma utilizza l'unità di controllo JDBC IBM Toolbox per Java per
// creare e popolare una tabella.
//
// Sintassi del comando:
//    JDBCPopulate system collectionName tableName
//
// Ad esempio,
//    JDBCPopulate MySystem MyLibrary MyTable
//
///////////////////////////////////////////////////////////////////////////////

import java.sql.*;

public class JDBCPopulate
{



        // Stringhe da aggiungere nella colonna WORD della tabella.
        private static final String words[]
        = { "One",      "Two",      "Three",    "Four",     "Five",
            "Six",      "Seven",    "Eight",    "Nine",     "Ten",
            "Eleven",   "Twelve",   "Thirteen", "Fourteen", "Fifteen",
            "Sixteen",  "Seventeen","Eighteen", "Nineteen", "Twenty" };



       public static void main (String[] parameters)
    {
              // Controllare i parametri di immissione.
                if (parameters.length != 3) {
                     System.out.println("");
                     System.out.println("Usage:");
                     System.out.println("");
            System.out.println("   JDBCPopulate system collectionName tableName");
                     System.out.println("");
                     System.out.println("");
                     System.out.println("For example:");
                     System.out.println("");
                     System.out.println("");
            System.out.println("   JDBCPopulate MySystem MyLibrary MyTable");
                     System.out.println("");
            return;
        }

                 String system = parameters[0];
```

```
        String collectionName   = parameters[1];
        String tableName        = parameters[2];

        Connection connection   = null;

        try {

                // Caricare l'unità di controllo JDBC IBM Toolbox per Java.
                DriverManager.registerDriver(new com.ibm.as400.access.AS400JDBCDriver());

    // Richiamare un collegamento al database. Poiché non viene fornito
                // un id utente o una parola d'ordine, verrà visualizzata una richiesta.
    //
                // Si noti che in questa sede viene fornito uno schema predefinito in modo tale che
                // non sia necessario qualificare il nome di tabella nelle
                // istruzioni SQL.
    //
    connection = DriverManager.getConnection ("jdbc:as400://" + system + "/" + collectionName);

                // Rilasciare la tabella se già esistente.
            try {
                        Statement dropTable = connection.createStatement ();
        dropTable.executeUpdate ("DROP TABLE " + tableName);
    }
                catch (SQLException e) {
                                    // Ignorare.
    }

                // Creare la tabella.
                Statement createTable = connection.createStatement ();
    createTable.executeUpdate ("CREATE TABLE " + tableName
        + " (I INTEGER, WORD VARCHAR(20), SQUARE INTEGER, "
        + " SQUAREROOT DOUBLE)");

    // Preparare un'istruzione per l'inserimento di righe. Poiché questa operazione viene
                // eseguita più volte, è più opportuno utilizzare una
                // PreparedStatement e contrassegni di parametro.
    PreparedStatement insert = connection.prepareStatement ("INSERT INTO "
        + tableName + " (I, WORD, SQUARE, SQUAREROOT) " + " VALUES (?, ?, ?, ?)");

                // Popolare la tabella.
                for (int i = 1; i <= words.length; ++i) {
                        insert.setInt (1, i);
                        insert.setString (2, words[i-1]);
                        insert.setInt (3, i*i);
                        insert.setDouble (4, Math.sqrt(i));
                        insert.executeUpdate ();
    }

                // Emettere un messaggio di completamento.
    System.out.println ("Table " + collectionName + "." + tableName + " has been populated.");
}

        catch (Exception e) {
                     System.out.println ();
    System.out.println ("ERROR: " + e.getMessage());
}

        finally {

                // Ripulire.
            try {
                        if (connection != null)
                                connection.close ();
    }
                catch (SQLException e) {
                                    // Ignorare.
```

```
            }
        }

              System.exit(0);
    }


}
```

Esempio: utilizzo di JDBCQuery per interrogare una tabella

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per informazioni legali importanti.

```
///////////////////////////////////////////////////////////////////////////////
//
// Esempio JDBCQuery. Questo programma utilizza l'unità di controllo JDBC IBM Toolbox per Java per
// interrogare una tabella ed emettere il relativo contenuto.
//
// Sintassi del comando:
//    JDBCQuery system collectionName tableName
//
// Ad esempio,
//    JDBCQuery MySystem qiws qcustcdt
//
///////////////////////////////////////////////////////////////////////////////

import java.sql.*;

public class JDBCQuery
{



        // Formattare una stringa in modo che abbia l'ampiezza specificata.
        private static String format (String s, int width)
    {
                String formattedString;

                // La stringa ha un'ampiezza inferiore a quella specificata,
                // quindi è necessario inserire degli spazi.
                if (s.length() < width) {
                        StringBuffer buffer = new StringBuffer (s);
                        for (int i = s.length(); i < width; ++i)
                buffer.append (" ");
                        formattedString = buffer.toString();
        }

                // Altrimenti, è necessario troncare la stringa.
                                else
                        formattedString = s.substring (0, width);

                return formattedString;
    }



       public static void main (String[] parameters)
    {
              // Controllare i parametri di immissione.
                if (parameters.length != 3) {
                     System.out.println("");
                     System.out.println("Usage:");
                     System.out.println("");
             System.out.println("   JDBCQuery system collectionName tableName");
                     System.out.println("");
                     System.out.println("");
```

```
            System.out.println("For example:");
            System.out.println("");
            System.out.println("");
    System.out.println("   JDBCQuery mySystem qiws qcustcdt");
            System.out.println("");
    return;
}

         String system = parameters[0];
        String collectionName   = parameters[1];
        String tableName        = parameters[2];

        Connection connection   = null;

        try {

                // Caricare l'unità di controllo JDBC IBM Toolbox per Java.
                DriverManager.registerDriver(new com.ibm.as400.access.AS400JDBCDriver());

    // Richiamare un collegamento al database. Poiché non viene fornito
                // un id utente o una parola d'ordine, verrà visualizzata una richiesta.
    connection = DriverManager.getConnection ("jdbc:as400://" + system);
                DatabaseMetaData dmd = connection.getMetaData ();

                // Eseguire l'interrogazione.
                Statement select = connection.createStatement ();
    ResultSet rs = select.executeQuery (
       "SELECT * FROM " + collectionName + dmd.getCatalogSeparator() + tableName);

    // Richiamare le informazioni sulla serie di risultati. Impostare l'ampiezza
                // della colonna sulla più lunga: lunghezza dell'etichetta
                // o lunghezza dei dati.
                ResultSetMetaData rsmd = rs.getMetaData ();
                int columnCount = rsmd.getColumnCount ();
                String[] columnLabels = new String[columnCount];
                int[] columnWidths = new int[columnCount];
                    for (int i = 1; i <= columnCount; ++i) {
                        columnLabels[i-1] = rsmd.getColumnLabel (i);
        columnWidths[i-1] = Math.max (columnLabels[i-1].length(), rsmd.getColumnDisplaySize (i));
    }

                // Emettere le intestazioni di colonna.
                    for (int i = 1; i <= columnCount; ++i) {
                        System.out.print (format (rsmd.getColumnLabel(i), columnWidths[i-1]));
                 System.out.print("   ");
    }
                    System.out.println ();

                // Emettere una riga con trattini.
                StringBuffer dashedLine;
                    for (int i = 1; i <= columnCount; ++i) {
                        for (int j = 1; j <= columnWidths[i-1]; ++j)
            System.out.print ("-");
                 System.out.print("   ");
    }
                    System.out.println ();

                // Ripetere per le righe nella serie di risultati ed emettere
                // le colonne relative ad ogni riga.
                while (rs.next ()) {
                        for (int i = 1; i <= columnCount; ++i) {
                                String value = rs.getString (i);
                                if (rs.wasNull ())
                value = "<null>";
                                System.out.print (format (value, columnWidths[i-1]));
                     System.out.print("   ");
        }
```

```
                                System.out.println ();
            }

        }

                catch (Exception e) {
                            System.out.println ();
            System.out.println ("ERROR: " + e.getMessage());
        }

                finally {

                        // Ripulire.
                    try {
                                if (connection != null)
                                        connection.close ();
            }
                            catch (SQLException e) {
                                                // Ignorare.
            }
        }

        System.exit (0);
    }


}
```

Esempio: utilizzo di JobList per elencare le informazioni di identificazione del lavoro

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per informazioni legali importanti.

```
///////////////////////////////////////////////////////////////////////////
//
// Questo programma è un esempio del supporto Job nell'IBM Toolbox
// per Java. Elenca le informazioni sull'identificativo del lavoro per uno specifico
// utente sul sistema.
//
// Sintassi del comando:
//    listJobs2 system userID password
//
///////////////////////////////////////////////////////////////////////////

import java.io.*;
import java.lang.*;
import java.util.*;
import com.ibm.as400.access.*;

public class listJobs2 extends Object
{

                                        // Creare un oggetto in caso si desideri chiamare
                                        // qualsiasi metodo non-static.
     public static void main(String[] parameters)
   {
           listJobs2 me = new listJobs2();
      me.Main(parameters);

      System.exit(0);
   }



   void Main(String[] parameters)
   {
```

```
              // Se non è stato specificato un sistema, visualizzare il testo di aiuto ed uscire.
if (parameters.length == 0)
{
   showHelp();
   return;
}

              // Assegnare i parametri alle variabili. Il
                                  // primo parametro si presume sia il nome del
                                  // sistema il secondo è l'ID utente ed il terzo
                                  // è una parola d'ordine.
      String systemName = parameters[0];
String userID   = null;
String password = null;

if (parameters.length > 1)
            userID = parameters[1].toUpperCase();

if (parameters.length >= 2)
            password = parameters[2].toUpperCase();

      System.out.println( " " );



                            try
{
                                  // Creare un oggetto AS400 utilizzando il nome di sistema
              // specificato dall'utente.  Impostare l'ID utente e
                                  // la parola d'ordine se specificati dall'utente.
   AS400 as400 = new AS400(parameters[0]);

   if (userID != null)
      as400.setUserId(userID);

   if (password != null)
      as400.setPassword(password);



   System.out.println("retrieving list ...     ");


              // Creare un oggetto jobList. Tale oggetto viene utilizzato
                                  // per richiamare l'elenco di lavori attivi sul server.
            JobList jobList = new JobList(as400);


                                    // Richiamare l'elenco di lavori attivi.
            Enumeration list = jobList.getJobs();


                                  // Per ogni lavoro nell'elenco ...
            while (list.hasMoreElements())
   {

              // Richiamare un oggetto dall'elenco. Se è stato specificato
                                  // un ID utente stampare le informazioni di identificazione
              // solo se l'utente del lavoro corrisponde all'ID utente. Se
                                  // non è stato specificato alcun ID utente stampare le informazioni
                                  // relative ad ogni lavoro sul sistema.
                  Job j = (Job) list.nextElement();

      if (userID != null)
      {
```

```
                    if (j.getUser().trim().equalsIgnoreCase(userID))
            {
               System.out.println(j.getName().trim() + "." +
                                  j.getUser().trim() + "." +
                                                                  j.getNumber());
            }
         }
                                 else
            System.out.println(j.getName().trim() + "." +
                               j.getUser().trim() + "." +
                                                               j.getNumber());
      }

   }
                                 catch (Exception e)
   {
      System.out.println("Unexpected error");
              e.printStackTrace();
   }
}



        // Visualizzare il testo di aiuto quando i parametri non sono corretti.
   void showHelp()
{
               System.out.println("");
               System.out.println("");
               System.out.println("");
      System.out.println("Parameters are not correct.  Command syntax is:");
               System.out.println("");
      System.out.println("   listJobs2 System UserID Password");
               System.out.println("");
               System.out.println("Where");
               System.out.println("");
      System.out.println("   System   = server to connect to");
      System.out.println("   UserID   = valid userID on that system ");
      System.out.println("   Password = password for the UserID (optional)");
               System.out.println("");
               System.out.println("For example:");
               System.out.println("");
      System.out.println("   listJobs2 MYAS400 JavaUser pwd1");
               System.out.println("");
               System.out.println("");
}
}
```

Esempio: utilizzo di JobList per richiamare un elenco di lavori

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per informazioni legali importanti.

```
///////////////////////////////////////////////////////////////////////////
//
// Questo programma è un esempio delle classi "job" in
// IBM Toolbox per Java. Richiama un elenco di lavori sul server
// ed emette lo stato del lavoro seguito dal relativo identificativo.
//
//
// Sintassi del comando:
//    listJobs system userID password
//
// (ID utente e parola d'ordine sono facoltativi)
//
///////////////////////////////////////////////////////////////////////////

import java.io.*;
import java.util.*;
```

```
import com.ibm.as400.access.*;

public class listJobs extends Object
{
   public static void main(String[] parameters)
   {
      listJobs me = new listJobs();
      me.Main(parameters);

      System.exit(0);
   }



   void Main(String[] parameters)
   {

      // Se non è stato specificato un sistema, visualizzare il testo di aiuto ed uscire.
      if (parameters.length == 0)
      {
         showHelp();
         return;
      }

      // Impostare i parametri dell'oggetto AS400. Il primo è il nome di sistema e deve essere
      // specificato dall'utente. Il secondo e il terzo sono facoltativi. Sono
      // l'ID utente e la parola d'ordine. Convertire l'ID utente e la parola d'ordine
      // in caratteri maiuscoli prima di impostarli sull'oggetto AS400.
      String userID   = null;
      String password = null;

      if (parameters.length > 1)
         userID = parameters[1].toUpperCase();

      if (parameters.length >= 2)
         password = parameters[2].toUpperCase();

      System.out.println(" ");

      try
      {

         // Creare un oggetto AS400 utilizzando il nome di sistema specificato dall'utente.
         AS400 as400 = new AS400(parameters[0]);

         // Se sono stati specificati un id utente e/o una parola d'ordine, impostarli
         // sull'oggetto AS400.
         if (userID != null)
            as400.setUserId(userID);

         if (password != null)
            as400.setPassword(password);

         // Creare un oggetto elenco lavori. Il parametro di immissione è l'AS400 dal quale
         // si desidera ricevere le informazioni sul lavoro.
         JobList jobList = new JobList(as400);

         // Richiamare un elenco di lavori in esecuzione sul server.
         Enumeration listOfJobs = jobList.getJobs();

         // Per ogni lavoro presente nell'elenco stampare le informazioni ad esso relative.
         while (listOfJobs.hasMoreElements())
         {
            printJobInfo((Job) listOfJobs.nextElement(), as400);
         }

      }
```

```
      catch (Exception e)
      {
         System.out.println("Unexpected error");
         System.out.println(e);
      }
   }



   void printJobInfo(Job job, AS400 as400)
   {

      // Creare i vari programmi di conversione necessari
            AS400Bin4 bin4Converter   = new AS400Bin4( );
            AS400Text text26Converter = new AS400Text(26, as400);
            AS400Text text16Converter = new AS400Text(16, as400);
            AS400Text text10Converter = new AS400Text(10, as400);
            AS400Text text8Converter  = new AS400Text(8,  as400);
            AS400Text text6Converter  = new AS400Text(6,  as400);
            AS400Text text4Converter  = new AS400Text(4,  as400);


      // Si dispone del nome/numero/ecc. del lavoro dalla richiesta dell'elenco. Ora,
      // effettuare una chiamata all'API del server per ottenere lo stato del lavoro.
      try
      {
         // Creare un oggetto chiamata a programma
         ProgramCall pgm = new ProgramCall(as400);

         // Il programma del server chiamato ha cinque parametri
         ProgramParameter[] parmlist = new ProgramParameter[5];

         // Il primo parametro è una schiera di byte che contiene i dati
         // Unicode.
Verrà assegnato un buffer di 1k per i dati di emissione.
         parmlist[0] = new ProgramParameter( 1024 );

         // Il secondo parametro indica la dimensione del buffer dei dati di emissione (1K).
         Integer iStatusLength = new Integer( 1024 );
         byte[]  statusLength = bin4Converter.toBytes( iStatusLength );
         parmlist[1] = new ProgramParameter( statusLength );

         // Il terzo parametro è il nome del formato dei dati.
         // Verrà utilizzato il formato JOBI0200 poiché dispone dello stato del lavoro.
         byte[] statusFormat = text8Converter.toBytes("JOBI0200");
         parmlist[2] = new ProgramParameter( statusFormat );

         // Il quarto parametro è il nome lavoro nel formato "nome utente numero".
         // Il nome deve contenere 10 caratteri, l'utente deve essere composto da 10 caratteri e
         // il numero deve essere composto da 6 caratteri. Verrà utilizzato un programma di conversione testo
         // per effettuare la conversione e l'inserimento.
         byte[] jobName = text26Converter.toBytes(job.getName());

         int    i       = text10Converter.toBytes(job.getUser(),
                                                  jobName,
                                                  10);

                i       = text6Converter.toBytes(job.getNumber(),
                                                 jobName,
                                                 20);

         parmlist[3] = new ProgramParameter( jobName );

         // L'ultimo parametro è l'identificativo lavoro. Verrà lasciato vuoto.
         byte[] jobID = text16Converter.toBytes("                ");
         parmlist[4] = new ProgramParameter( jobID );
```

```
        // Eseguire il programma.
        if (pgm.run( "/QSYS.LIB/QUSRJOBI.PGM", parmlist )==false)
        {
           // se il programma ha dato esito negativo visualizzare il messaggio di errore.
           AS400Message[] msgList = pgm.getMessageList();
           System.out.println(msgList[0].getText());
        }
        else
        {
           // in un altro contesto il programma ha funzionato. Emettere lo stato seguito da
           // jobName.user.jobID
           byte[] as400Data = parmlist[0].getOutputData();
           System.out.print("  " + text4Converter.toObject(as400Data, 107) + "  ");

           System.out.println(job.getName().trim() + "." +
                              job.getUser().trim() + "." +
                              job.getNumber() + "   ");
        }

      }
      catch (Exception e)
      {
         System.out.println(e);
      }

   }



      // Visualizzare il testo di aiuto quando i parametri non sono corretti.
   void showHelp()
   {
         System.out.println("");
         System.out.println("");
         System.out.println("");
         System.out.println("Parameters are not correct.  Command syntax is:");
         System.out.println("");
         System.out.println("   listJobs System UserID Password");
         System.out.println("");
         System.out.println("Where");
         System.out.println("");
         System.out.println("   System   = server to connect to");
         System.out.println("   UserID   = valid userID on that system (optional)");
         System.out.println("   Password = password for the UserID (optional)");
         System.out.println("");
         System.out.println("For example:");
         System.out.println("");
         System.out.println("   listJobs MYAS400 JavaUser pwd1");
         System.out.println("");
         System.out.println("");
   }
}
```

Esempio: utilizzo di JobLog per visualizzare messaggi nella registrazione lavori

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per informazioni legali importanti.

```
////////////////////////////////////////////////////////////////////////////
//
// Questo programma è un esempio della funzione di registrazione lavori di
// IBM Toolbox per Java. Visualizzerà i messaggi nella registrazione
// lavori per il lavoro che appartiene all'utente corrente.
//
// Sintassi del comando:
//    jobLogExample system userID password
//
```

```
// (La parola d'ordine è facoltativa)
//
//////////////////////////////////////////////////////////////////////////

import java.lang.*;
import java.util.*;
import com.ibm.as400.access.*;

public class jobLogExample
{


        public static void main (String[] args)
    {
        // Se non sono stati specificati un sistema ed un utente, visualizzare il testo di aiuto ed uscire.
                if (args.length < 2)
        {
            System.out.println("Usage:  jobLogExample system userid <password>");
                        return;
        }

        String userID   = null;

                                    try
        {
            // Creare un oggetto AS400.
Il nome di sistema è passato
                        // come primo argomento della riga comandi.
Se l'ID utente e la
                        // parola d'ordine sono stati passati sulla riga comandi,
                        // impostare anche tali valori.
                        AS400 system = new AS400 (args[0]);

            if (args.length > 1)
            {
                                userID = args[1];
                                system.setUserId(userID);
            }

            if (args.length > 2)
                                system.setPassword(args[2]);


            // Creare un oggetto elenco lavori. Yale oggetto verrà utilizzato per richiamare
            // l'elenco di lavori attivi sul sistema. Una volta richiamato l'elenco
                        // di lavori, il programma individuerà il lavoro relativo
                        // all'utente corrente.
                        JobList jobList = new JobList(system);

                           // Richiamare l'elenco di lavori attivi sull'AS/400
                      Enumeration list = jobList.getJobs();

                           boolean Continue = true;

                           // Esaminare l'elenco per individuare un lavoro relativo all'utente corrente.
                           while (list.hasMoreElements() && Continue)
            {
                              Job j = (Job) list.nextElement();

                            if (j.getUser().trim().equalsIgnoreCase(userID))
               {
                  // E' stato rilevato un lavoro che corrisponde all'utente corrente. Creare
                  JobLog jlog = new JobLog(system, j.getName(), j.getUser(), j.getNumber());

                                    // Enumerare i messaggi nella registrazione lavori e quindi stamparli.
                               Enumeration messageList = jlog.getMessages();
```

```
                        while (messageList.hasMoreElements())
                {
                                AS400Message message = (AS400Message) messageList.nextElement();
                        System.out.println(message.getText());
                }

                // E' stato trovato un lavoro corrispondente all'utente corrente
                // quindi è possibile uscire.
                                Continue = false;
              }
            }
        }
                                catch (Exception e)
        {
                        System.out.println ("Error: " + e.getMessage ());
        }

        System.exit(0);
    }

}
```

Esempio: creazione di file di spool

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per informazioni legali importanti.

```
///////////////////////////////////////////////////////////////////////////
//
// Esempio che illustra la creazione di un file di spool su un server da un flusso di immissione.
//
///////////////////////////////////////////////////////////////////////////

import java.io.*;
import java.util.*;

import com.ibm.as400.access.*;

class NPExampleCreateSplf
{

// metodo per creare il file di spool sul server specificato, nella coda di emissione
// indicata dal flusso di immissione fornito.
public SpooledFile createSpooledFile(AS400 system, OutputQueue outputQueue, InputStream in)
{
        SpooledFile spooledFile = null;
                            try
    {
                byte[] buf = new byte[2048];
                int bytesRead;
                SpooledFileOutputStream out;
                    PrintParameterList parms = new PrintParameterList();

                // creare una PrintParameterList con i valori che si desidera
                // sostituire dal file di stampa predefinito...si sostituirà
                // la coda di emissione ed il valore delle copie.
                parms.setParameter(PrintObject.ATTR_COPIES, 4);
                if (outputQueue != null)
        {
                      parms.setParameter(PrintObject.ATTR_OUTPUT_QUEUE, outputQueue.getPath());
        }
                out = new SpooledFileOutputStream(system,
                                                                                parms,
                                                                                null,
                                                                                null);

                // leggere nel flusso di immissione fino alla fine del flusso, passando tutti i dati
                // al flusso di emissione del file di spool.
```

```
                do
        {
                    bytesRead = in.read(buf);
                    if (bytesRead != -1)
            {
                out.write(buf, 0, bytesRead);
            }
                } while (bytesRead != -1);

                out.close();  // chiudere il file di spool

                spooledFile = out.getSpooledFile();   // richiamare un riferimento al nuovo file di spool

    }
                            catch (Exception e)
    {
                //...gestire l'eccezione...
    }
        return spooledFile;
}

}
```

Esempio: creazione di file di spool SCS

Questo esempio utilizza la classe SCS3812Writer per creare un flusso di dati SCS e inserirlo in un file di spool sul server.

Quest'applicazione può utilizzare i seguenti argomenti o i valori predefiniti stabiliti:

- Nome del server per ricevere il file di spool.
- Nome della coda esterna del server che riceverà il file di spool.

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per informazioni legali importanti.

```
///////////////////////////////////////////////////////////////////////////
//
// Questo sorgente è un esempio di "SCS3812Writer" di IBM Toolbox per Java.
//
///////////////////////////////////////////////////////////////////////////

import com.ibm.as400.access.*;

class NPExampleCreateSCSSplf
{
    private static final String DEFAULT_SYSTEM = new String("RCHAS1");
    private static final String DEFAULT_OUTQ = new String("/QSYS.LIB/QUSRSYS.LIB/PRT01.OUTQ");

        public static void main(String [] args)
    {
                                    try
        {
                    AS400 system;
                SpooledFileOutputStream out;
                    PrintParameterList parms = new PrintParameterList();
                    SCS3812Writer scsWtr;

                    // Elaborare gli argomenti.
            if (args.length >= 1)
            {
                            system = new AS400(args[0]);    // Creare un oggetto AS400
                    } else {
                            system = new AS400(DEFAULT_SYSTEM);
            }

            if (args.length >= 2)               // Set the outq
```

```
    {
                        parms.setParameter(PrintObject.ATTR_OUTPUT_QUEUE, args[1]);
                } else {
                        parms.setParameter(PrintObject.ATTR_OUTPUT_QUEUE, DEFAULT_OUTQ);
    }

                out = new SpooledFileOutputStream(system, parms, null, null);

                scsWtr = new SCS3812Writer(out, 37);

                // Scrivere il contenuto del file di spool.
                scsWtr.setLeftMargin(1.0);
                scsWtr.absoluteVerticalPosition(6);
                scsWtr.setFont(scsWtr.FONT_COURIER_BOLD_5);
    scsWtr.write("          Java Printing");
                scsWtr.newLine();
                scsWtr.newLine();
                scsWtr.setCPI(10);
    scsWtr.write("This document was created using the IBM Toolbox for Java.");
                scsWtr.newLine();
    scsWtr.write("The rest of this document shows some of the things that");
                scsWtr.newLine();
    scsWtr.write("can be done with the SCS3812Writer class.");
                scsWtr.newLine();
                scsWtr.newLine();
    scsWtr.setUnderline(true); scsWtr.write("Setting fonts:"); scsWtr.setUnderline(false);
                scsWtr.newLine();
    scsWtr.setFont(scsWtr.FONT_COURIER_10); scsWtr.write("Courier font ");
    scsWtr.setFont(scsWtr.FONT_COURIER_BOLD_10); scsWtr.write(" Courier bold font ");
    scsWtr.setFont(scsWtr.FONT_COURIER_ITALIC_10); scsWtr.write(" Courier italic font ");
                scsWtr.newLine();
    scsWtr.setBold(true); scsWtr.write("Courier bold italic font ");
                scsWtr.setBold(false);
                scsWtr.setCPI(10);
                scsWtr.newLine();
                scsWtr.newLine();
    scsWtr.setUnderline(true); scsWtr.write("Lines per inch:"); scsWtr.setUnderline(false);
                scsWtr.newLine();
    scsWtr.write("The following lines should print at 8 lines per inch.");
                scsWtr.newLine();
                scsWtr.newLine();
                scsWtr.setLPI(8);
    scsWtr.write("Line one"); scsWtr.newLine();
    scsWtr.write("Line two"); scsWtr.newLine();
    scsWtr.write("Line three"); scsWtr.newLine();
    scsWtr.write("Line four"); scsWtr.newLine();
    scsWtr.write("Line five"); scsWtr.newLine();
    scsWtr.write("Line six"); scsWtr.newLine();
    scsWtr.write("Line seven"); scsWtr.newLine();
    scsWtr.write("Line eight"); scsWtr.newLine();
                scsWtr.endPage();
                scsWtr.setLPI(6);
                scsWtr.setSourceDrawer(1);
                scsWtr.setTextOrientation(0);
                scsWtr.absoluteVerticalPosition(6);
    scsWtr.write("This page should print in portrait orientation from drawer 1.");
                scsWtr.endPage();
                scsWtr.setSourceDrawer(2);
                scsWtr.setTextOrientation(90);
                scsWtr.absoluteVerticalPosition(6);
    scsWtr.write("This page should print in landscape orientation from drawer 2.");
                scsWtr.endPage();
                scsWtr.close();
    System.out.println("Sample spool file created.");
            System.exit(0);
}
                            catch (Exception e)
```

```
        {
                        // Gestire l'errore.
            System.out.println("Exception occured while creating spooled file. " + e);
                    System.exit(0);
        }
    }
}
```

Esempio: lettura dei file di spool

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per informazioni legali importanti.

```
///////////////////////////////////////////////////////////////////////
//
// Esempio che legge un file di spool del server esistente.
//
// Questo sorgente è un esempio di "PrintObjectInputStream" di IBM Toolbox per Java.
//
///////////////////////////////////////////////////////////////////////
                try{
                byte[] buf = new byte[2048];
                int bytesRead;
                AS400 sys = new AS400();
                SpooledFile splf = new SpooledFile( sys,          // AS400
                                                    "MICR",       // nome splf
                                                    17,           // numero splf
                                                    "QPRTJOB",    // nome lavoro
                                                    "QUSER",      // utente lavoro
                                                    "020791" );   // numero lavoro

                // aprire il file di spool per la lettura e richiamare il flusso di immissione
                // per leggerlo.
                InputStream in = splf.getInputStream(null);

                    do
        {
                        // leggere fino a buf.length byte di dati spool grezzi nel
            // buffer. Verranno restituiti i byte effettivamente letti.
                        // I dati formeranno un flusso di dati di stampa binari che rappresenta il
                        // contenuto del file di spool.
                        bytesRead = in.read( buf );
                        if( bytesRead != -1 )
            {
                                // elaborare i dati del file di spool.
               System.out.println( "Read " + bytesRead + " bytes" );
            }
                } while( bytesRead != -1 );

                in.close();
    }
        catch( Exception e )
    {
                // eccezione
    }
```

Esempio: lettura e trasformazione dei file di spool

Gli esempi che seguono mostrano come impostare un PrintParameterList per ottenere trasformazioni diverse durante la lettura dei dati del file di spool. Nei segmenti di codice che seguono, presupporre che un file di spool esista già su un server, e che il metodo *createSpooledFile()* crei un'istanza della classe SpooledFile che rappresenta il file di spool.

Esempio di PrintObjectPageInputStream

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

L'esempio che segue mostra come creare un oggetto PrintObjectPageInputStream per la lettura di pagine dei dati formattati come immagini GIF. In questo caso, ogni pagina dal file di spool sarà trasformata in un'immagine GIF. Un oggetto di personalizzazione della stazione di lavoro GIF viene utilizzato per specificare la trasformazione dei dati.

```
// Creare un file di spool
SpooledFile splF = createSpooledFile();

// Impostare un elenco di parametri di stampa
PrintParameterList printParms = new PrintParameterList();
printParms.setParameter(PrintObject.ATTR_WORKSTATION_CUST_OBJECT, "/QSYS.LIB/QWPGIF.WSCST");
printParms.setParameter(PrintObject.ATTR_MFGTYPE, "*WSCST");

// Creare un flusso di immissione pagina dal file di spool
PrintObjectPageInputStream is = splF.getPageInputStream(printParms);
```

Esempio di PrintObjectTransformedInputStream

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

L'esempio che segue mostra come creare un oggetto PrintObjectTransformedInputStream per la lettura dei dati formattati come TIFF. Un oggetto di personalizzazione della stazione di lavoro TIFF (compressione G4) viene utilizzato per specificare la trasformazione dei dati.

```
// Creare un file di spool
SpooledFile splF = createSpooledFile();

// Impostare un elenco di parametri di stampa
PrintParameterList printParms = new PrintParameterList();
printParms.setParameter(PrintObject.ATTR_WORKSTATION_CUST_OBJECT, "/QSYS.LIB/QWPTIFFG4.WSCST");
printParms.setParameter(PrintObject.ATTR_MFGTYPE, "*WSCST");

// Creare un flusso di immissione convertito dal file di spool
PrintObjectTransformedInputStream is = splF.getTransformedInputStream(printParms);
```

Esempio di PrintObjectTransformedInputStream che utilizza il tipo e il modello del produttore

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

L'esempio che segue mostra come creare un oggetto PrintObjectTransformedInputStream per la lettura dei dati formattati per l'emissione su una stampante ASCII. Un tipo e modello di produttore di *HP4 viene utilizzato per specificare la trasformazione dei dati.

```
// Creare un file di spool
SpooledFile splF = createSpooledFile();

// Impostare un elenco di parametri di stampa
PrintParameterList printParms = new PrintParameterList();
printParms.setParameter(PrintObject.ATTR_MFGTYPE, "*HP4");

// Creare un flusso di immissione convertito dal file di spool
PrintObjectTransformedInputStream is = splF.getTransformedInputStream(printParms);
```

Esempio: elenco dei file di spool in modalità asincrona (utilizzando i listener)

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per informazioni legali importanti.

```
////////////////////////////////////////////////////////////////////////
//
// Esempio che mostra l'elenco di tutti i file di spool su un server in modo asincrono utilizzando
// l'interfaccia PrintObjectListListener per ottenere un feedback mentre viene creato l'elenco.
// L'elenco in modo asincrono consente al chiamante di avviare l'elaborazione dell'elenco di oggetti
// prima che l'intero elenco sia creato per un tempo di risposta percepita più rapido
// per l'utente.
//
```

```
////////////////////////////////////////////////////////////////////////

import com.ibm.as400.access.AS400;
import com.ibm.as400.access.SpooledFileList;
import com.ibm.as400.access.SpooledFile;
import com.ibm.as400.access.ExtendedIllegalStateException;
import com.ibm.as400.access.PrintObjectListListener;
import com.ibm.as400.access.PrintObjectListEvent;

public class NPExampleListSplfAsynch extends Object implements PrintObjectListListener
{
        private AS400 system_;
        private boolean fListError;
        private boolean fListClosed;
        private boolean fListCompleted;
        private Exception listException;
        private int listObjectCount;

        public NPExampleListSplfAsynch(AS400 system)
    {
                system_ = system;
    }

    // elencare tutti i file di spool sul server in modo asincrono utilizzando un listener
        public void listSpooledFiles()
    {
                fListError = false;
                fListClosed = false;
                fListCompleted = false;
                listException = null;
                listObjectCount = 0;

                                try
        {
                        String strSpooledFileName;
                        boolean fCompleted = false;
                        int listed = 0, size;

                        if( system_ == null )
            {
                        system_ = new AS400();
            }

            System.out.println(" Now receiving all spooled files Asynchronously using a listener");

                        SpooledFileList splfList = new SpooledFileList(system_);

                        // impostare i filtri, tutti gli utenti, su tutte le code
            splfList.setUserFilter("*ALL");
            splfList.setQueueFilter("/QSYS.LIB/%ALL%.LIB/%ALL%.OUTQ");

                        // aggiungere il listener.
                        splfList.addPrintObjectListListener(this);

                        // aprire l'elenco, viene immediatamente restituito openAsynchronously
                        splfList.openAsynchronously();

                        do
            {
                                // attendere che l'elenco contenga almeno 25 oggetti o che venga eseguita
                                waitForWakeUp();

                                fCompleted = splfList.isCompleted();
                                size = splfList.size();

                                // emettere i nomi di tutti gli oggetti aggiunti all'elenco
                                // dall'ultima attivazione
```

```
                        while (listed < size)
            {
                                if (fListError)
                {
                    System.out.println(" Exception on list - " + listException);
                                    break;
                }

                                if (fListClosed)
                {
                    System.out.println(" The list was closed before it completed!");
                                    break;
                }

                                    SpooledFile splf = (SpooledFile)splfList.getObject(listed++);
                            if (splf != null)
                {
                    // emettere questo nome di file di spool
                    strSpooledFileName = splf.getStringAttribute(SpooledFile.ATTR_SPOOLFILE);
                    System.out.println(" spooled file = " + strSpooledFileName);
                }
            }

                    } while (!fCompleted);

                    // ripulire una volta terminato con l'elenco
                    splfList.close();
                    splfList.removePrintObjectListListener(this);
    }

            catch( ExtendedIllegalStateException e )
    {
        System.out.println(" The list was closed before it completed!");
    }

        catch( Exception e )
    {
                    // ...gestire qualsiasi altra eccezione...
        e.printStackTrace();
    }

}

    // Questo è il punto in cui il sottoprocesso di foreground attende l'attivazione da parte del
    // del sottoprocesso di background quando l'elenco viene aggiornato o termina.
private synchronized void waitForWakeUp() throws InterruptedException
{
            // non disattivarsi se il listener informa che l'elenco è stato eseguito
            if (!fListCompleted)
    {
                    wait();
    }
}

    // I seguenti metodi implementano l'interfaccia PrintObjectListListener

    // Questo metodo è richiamato quando si chiude l'elenco.
    public void listClosed(PrintObjectListEvent event)
{
    System.out.println("*****The list was closed*****");
            fListClosed = true;
            synchronized(this)
    {
                    // Impostare l'indicatore in modo da indicare che l'elenco è stato
                    // completato ed attivare il sottoprocesso di foreground.
                    fListCompleted = true;
                        notifyAll();
```

```
    }
}

    // Questo metodo viene richiamato quando l'elenco è stato completato.
    public void listCompleted(PrintObjectListEvent event)
{
    System.out.println("*****The list has completed*****");
                synchronized (this)
    {
                    // Impostare l'indicatore in modo da indicare che l'elenco è stato
                    // completato ed attivare il sottoprocesso di foreground.
                    fListCompleted = true;
                        notifyAll();
    }
}

    // Questo metodo viene richiamato se si verifica un errore durante il recupero
    // dell'elenco.
    public void listErrorOccurred(PrintObjectListEvent event)
{
    System.out.println("*****The list had an error*****");
            fListError = true;
            listException = event.getException();
            synchronized(this)
    {
                    // Impostare l'indicatore in modo da indicare che l'elenco è stato
                    // completato ed attivare il sottoprocesso di foreground.
                    fListCompleted = true;
                        notifyAll();
    }
}

    // Questo metodo viene richiamato quando si apre l'elenco.
    public void listOpened(PrintObjectListEvent event)
{
    System.out.println("*****The list was opened*****");
            listObjectCount = 0;
}

    // Questo metodo viene richiamato quando un oggetto viene aggiunto all'elenco.
    public void listObjectAdded(PrintObjectListEvent event)
{
            // ogni 25 oggetti verrà attivato il sottoprocesso
            // di foreground per richiamare gli ultimi oggetti...
            if( (++listObjectCount % 25) == 0 )
    {
        System.out.println("*****25 more objects added to the list*****");
                    synchronized (this)
        {
                            // attivare il sottoprocesso di foreground
                            notifyAll();
        }
    }
}

    public static void main( String args[] )
{
            NPExampleListSplfAsynch list = new NPExampleListSplfAsynch(new AS400());
            try{
                    list.listSpooledFiles();
    }
        catch( Exception e )
    {
        e.printStackTrace();
    }
```

```
        System.exit(0);
    }

}
```

Esempio: elenco dei file di spool in modalità asincrona (senza utilizzare i listener)

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per informazioni legali importanti.

```
///////////////////////////////////////////////////////////////////////
//
// Esempio che mostra l'elencazione di tutti i file di spool su un sistema in modo Asincrono senza
// utilizzare l'interfaccia PrintObjectListListener. Dopo aver aperto l'elenco. il chiamante
// può svolgere del lavoro supplementare prima di attendere il completamento dell'elenco.
//
///////////////////////////////////////////////////////////////////////
//
// Questo sorgente è un esempio di "PrintObjectList" di IBM Toolbox per Java.
//
///////////////////////////////////////////////////////////////////////

import java.util.Enumeration;

import com.ibm.as400.access.AS400;
import com.ibm.as400.access.SpooledFileList;
import com.ibm.as400.access.SpooledFile;

public class NPExampleListSplfAsynch2 extends Object
{
        private AS400 system_;

        public NPExampleListSplfAsynch2(AS400 system)
    {
                system_ = system;
    }

        // elencare tutti i file di spool sul sistema in modo asincrono
        public void listSpooledFiles()
    {
                                    try
        {
                        String strSpooledFileName;
                        int listed, size;

                        if( system_ == null )
            {
                        system_ = new AS400();
            }

            System.out.println(
               "Now receiving all spooled files Asynchronously without using a listener");

                        SpooledFileList splfList = new SpooledFileList(system_);

                        // impostare i filtri, tutti gli utenti, su tutte le code
            splfList.setUserFilter("*ALL");
            splfList.setQueueFilter("/QSYS.LIB/%ALL%.LIB/%ALL%.OUTQ");

                        // aprire l'elenco, openAsynchronously() viene restituito immediatamente
                        // non è stato aggiunto alcun listener...
                        splfList.openAsynchronously();

            System.out.println(" Do some processing before waiting...");

                        // ... effettuare l'elaborazione a questo punto durante la creazione dell'elenco....

            System.out.println(" Now wait for list to complete.");
```

```
                        // attendere il completamento dell'elenco
                        splfList.waitForListToComplete();

                        Enumeration enum = splfList.getObjects();

            // emettere il nome di tutti gli oggetti nell'elenco
                        while( enum.hasMoreElements() )
            {
                SpooledFile splf = (SpooledFile)enum.nextElement();
                                if (splf != null)
                {
                    // emettere il nome di questo file di spool
                    strSpooledFileName = splf.getStringAttribute(SpooledFile.ATTR_SPOOLFILE);
                    System.out.println(" spooled file = " + strSpooledFileName);
                }
            }
                        // ripulire una volta terminato con l'elenco
                        splfList.close();
        }

            catch( Exception e )
        {
                        // ...gestire qualsiasi eccezione...
            e.printStackTrace();
        }
    }

        public static void main( String args[] )
    {
                NPExampleListSplfAsynch2 list = new NPExampleListSplfAsynch2(new AS400());
                try{
                        list.listSpooledFiles();
        }
            catch( Exception e )
        {
            e.printStackTrace();
        }
        System.exit(0);
    }
}
```

Esempio: elenco dei file di spool in modalità sincrona

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per informazioni legali importanti.

```
///////////////////////////////////////////////////////////////////////////
//
// Esempio che mostra l'elenco di tutti i file di spool su un server in modo sincrono.
// L'elenco in modo sincrono non ritorna al chiamante fino a quando non viene creato
// completamente l'elenco. L'utente percepisce un tempo di risposta più lento rispetto all'elenco in modo sincrono.
//
///////////////////////////////////////////////////////////////////////////
//
// Questo sorgente è un esempio di "PrintObjectList" di IBM Toolbox per Java.
//
///////////////////////////////////////////////////////////////////////////

import java.util.Enumeration;

import com.ibm.as400.access.AS400;
import com.ibm.as400.access.SpooledFileList;
import com.ibm.as400.access.SpooledFile;

public class NPExampleListSplfSynch
{
        private AS400 system_ = new AS400();
```

```
    public NPExampleListSplfSynch(AS400 system)
{
        system_ = system;
}

    public void listSpooledFiles()
{
        try{
                String strSpooledFileName;

                if( system_ == null )
        {
                system_ = new AS400();
        }

        System.out.println(" Now receiving all spooled files Synchronously");

                SpooledFileList splfList = new SpooledFileList( system_ );

                // impostare i filtri, tutti gli utenti, su tutte le code
        splfList.setUserFilter("*ALL");
        splfList.setQueueFilter("/QSYS.LIB/%ALL%.LIB/%ALL%.OUTQ");

                // elenco aperto, openSynchronously() restituito al completamento dell'elenco.
                splfList.openSynchronously();
                Enumeration enum = splfList.getObjects();

                while( enum.hasMoreElements() )
        {
            SpooledFile splf = (SpooledFile)enum.nextElement();
                        if ( splf != null )
            {
                // emettere il nome di questo file di spool
                strSpooledFileName = splf.getStringAttribute(SpooledFile.ATTR_SPOOLFILE);
                System.out.println(" spooled file = " + strSpooledFileName);
            }
        }
                // ripulire una volta terminato con l'elenco
                splfList.close();
    }
        catch( Exception e )
    {
                // ...gestire qualsiasi eccezione...
        e.printStackTrace();
    }
}

    public static void main( String args[] )
{
        NPExampleListSplfSynch list = new NPExampleListSplfSynch(new AS400());
        try{
                list.listSpooledFiles();
    }
        catch( Exception e )
    {
        e.printStackTrace();
    }
    System.exit(0);
}
}
```

Esempio: utilizzo di ProgramCall

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per informazioni legali importanti.

```
////////////////////////////////////////////////////////////////////////
//
// Esempio di chiamata del programma. Tale programma richiama il programma server QWCRSSTS
// per reperire lo stato del sistema.
//
// Sintassi del comando:
//    PCSystemStatusExample system
//
// Questo sorgente è un esempio di "ProgramCall" di IBM Toolbox per Java.
//
////////////////////////////////////////////////////////////////////////



import java.io.*;
import java.util.*;
import java.math.*;
import java.lang.Thread.*;
import com.ibm.as400.access.*;

public class PCSystemStatusExample extends Object
{
     public static void main(String[] parameters)
   {
            System.out.println( " " );



            // se non è stato specificato un sistema, visualizzare il testo di aiuto ed uscire.

      if (parameters.length >= 1)
      {

                                    try
         {
                        // Creare un oggetto AS400 per il server che contiene il
                                   // Java.
Supponiamo che il primo parametro sia il nome di sistema.

                        AS400 as400 = new AS400(parameters[0]);



                        // Creare il percorso al programma.

            QSYSObjectPathName programName = new QSYSObjectPathName("QSYS", "QWCRSSTS", "PGM");



            // Creare l'oggetto ProgramCall. Associare l'oggetto all'oggetto
                        // AS400 che rappresenta il server da cui si richiama lo stato.

                        ProgramCall getSystemStatus = new ProgramCall(as400);



            // Creare l'elenco parametri programma. Tale programma dispone di cinque
                        // parametri che verranno aggiunti all'elenco.

                        ProgramParameter[] parmlist = new ProgramParameter[5];



            // Il programma server restituisce i dati nel parametro 1. Si tratta di un parametro di
                                   // emissione.
Assegnare 64 byte per questo parametro.
```

```
                        parmlist[0] = new ProgramParameter( 64 );



            // Il parametro 2 corrisponde alla dimensione del buffer del parm 1. E' un parametro di
            // immissione numerico.
Impostare il relativo valore su 64, convertirlo nel formato server,
                        // quindi aggiungere il parametro all'elenco di parametri.

                        AS400Bin4 bin4 = new AS400Bin4( );
                        Integer iStatusLength = new Integer( 64 );
                        byte[]  statusLength = bin4.toBytes( iStatusLength );
            parmlist[1] = new ProgramParameter( statusLength );



            // Il parametro 3 corrisponde al parametro formato stato. E' un parametro di
            // immissione stringa.
Impostare il valore di stringa, convertirlo nel formato server,
                        // quindi aggiungere il parametro all'elenco di parametri.

                        AS400Text text1 = new AS400Text(8, as400);
            byte[] statusFormat = text1.toBytes("SSTS0200");
                        parmlist[2] = new ProgramParameter( statusFormat );



            // Il parametro 4 corrisponde al parametro statistiche-reimpostazione. E' un parametro di
            // immissione stringa.
Impostare il valore di stringa, convertirlo nel formato server,
                        // quindi aggiungere il parametro all'elenco di parametri.

                        AS400Text text3 = new AS400Text(10, as400);
            byte[] resetStats = text3.toBytes("*NO       ");
                        parmlist[3] = new ProgramParameter( resetStats );



            // Il parametro 5 è il parametro delle informazioni sull'errore. E' un parametro
            // di immissione/emissione.
Aggiungerlo all'elenco parametri.

                        byte[] errorInfo = new byte[32];
                        parmlist[4] = new ProgramParameter( errorInfo, 0 );



                        // Impostare il programma da chiamare e l'elenco di parametri sull'oggetto
                        // chiamata al programma.

                        getSystemStatus.setProgram(programName.getPath(), parmlist );



            // Eseguire il programma e poi rimanere inattivi. Il programma viene eseguito due volte perché
            // la prima serie di risultati non è attendibile. Se si elimina la prima
                        // serie di risultati e si esegue nuovamente il comando dopo cinque secondi il
                        // numero sarà più accurato.

                        getSystemStatus.run();
                        Thread.sleep(5000);



                        // Eseguire il programma
```

```
            if (getSystemStatus.run()!=true)
{
                    // Se il programma non è stato eseguito richiamare l'elenco di messaggi di errore
   // dall'oggetto programma e visualizzare i messaggi. L'errore
                    // dovrebbe essere del tipo programma-non-trovato o non-autorizzato
                    // al programma.

                    AS400Message[] msgList = getSystemStatus.getMessageList();

   System.out.println("The program did not run.  Server messages:");

                    for (int i=0; i<msgList.length; i++)
   {
                            System.out.println(msgList[i].getText());
   }
}


            // Altrimenti il programma è stato eseguito.

                         else
{
   // Creare un programma di conversione da server a numerico Java. Tale programma di conversione
                    // verrà utilizzato nella seguente sezione per convertire l'emissione
                    // numerica dal formato server al formato Java.

                    AS400Bin4 as400Int = new AS400Bin4( );



   // Richiamare i risultati del programma. I dati di emissione si trovano in
                    // una schiera di byte nel primo parametro.

                    byte[] as400Data = parmlist[0].getOutputData();



                    // L'utilizzo CPU è un campo numerico che inizia al byte
   // 32 del buffer di emissione. Convertire tale numero dal
                    // formato server al formato Java ed emettere il numero.

                    Integer cpuUtil = (Integer)as400Int.toObject( as400Data, 32 );
                    cpuUtil = new Integer(cpuUtil.intValue()/10);
   System.out.print("CPU Utilization:  ");
                    System.out.print(cpuUtil);
   System.out.println("%");



                    // L'utilizzo DASD è un campo numerico che inizia al byte
   // 52 del buffer di emissione. Convertire tale numero dal
                    // formato server al formato Java ed emettere il numero.

                    Integer dasdUtil = (Integer)as400Int.toObject( as400Data, 52 );
                    dasdUtil = new Integer(dasdUtil.intValue()/10000);
   System.out.print("Dasd Utilization: ");
                    System.out.print(dasdUtil);
   System.out.println("%");



                    // Il numero di lavori è un campo numerico che inizia al byte
   // 36 del buffer di emissione. Convertire tale numero dal
                    // formato server al formato Java ed emettere il numero.
```

```
                        Integer nj = (Integer)as400Int.toObject( as400Data, 36 );
            System.out.print("Active jobs:      ");
                        System.out.println(nj);

         }

            // Questo programma ha terminato l'esecuzione del programma quindi scollegarsi dal
            // dal server del comando sul server. La chiamata al programma e la chiamata al comando,
            // utilizzano lo stesso server sul server.

                    as400.disconnectService(AS400.COMMAND);
      }
                                catch (Exception e)
      {
         // Se una qualsiasi delle operazioni riportate sopra ha dato esito negativo
         // considerare non riuscito il programma ed emettere l'eccezione.

         System.out.println("Program call failed");
                 System.out.println(e);
      }
   }

   // Visualizzare il testo di aiuto quando i parametri non sono corretti.

                        else
   {
               System.out.println("");
               System.out.println("");
               System.out.println("");
      System.out.println("Parameters are not correct.  Command syntax is:");
               System.out.println("");
      System.out.println("   PCSystemStatusExample myServer");
               System.out.println("");
               System.out.println("Where");
               System.out.println("");
      System.out.println("   myServer = get status of this server ");
               System.out.println("");
               System.out.println("For example:");
               System.out.println("");
      System.out.println("   PCSystemStatusExample mySystem");
               System.out.println("");
               System.out.println("");
   }

   System.exit(0);
  }
}
```

Esempio: utilizzo delle classi di accesso al livello record

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
////////////////////////////////////////////////////////////////////////////
//
// Esempio di accesso a livello record.
Questo programma richiederà all'utente
// il nome del server ed il file da visualizzare.  Il file deve esistere
// e contenere record.  Ogni record nel file verrà visualizzato
// in System.out.
//
// Sintassi di chiamata: java RLSequentialAccessExample
//
// Questo sorgente è un esempio di "RecordLevelAccess" IBM Toolbox per Java
//
////////////////////////////////////////////////////////////////////////////
```

```
import java.io.*;
import java.util.*;
import com.ibm.as400.access.*;

public class RLSequentialAccessExample
{
      public static void main(String[] parmeters)
   {
            // Creato un programma di lettura per richiamare l'immissione dall'utente
            BufferedReader inputStream = new BufferedReader(new InputStreamReader(System.in),1);

          // Dichiarare variabili per contenere il nome di sistema, i nomi libreria, file e membro
      String systemName = "";
      String library = "";
      String file = "";
      String member = "";

            // Richiamare il nome di sistema ed il file da visualizzare dall'utente
          System.out.println();
                                try
      {
        System.out.print("System name: ");
              systemName = inputStream.readLine();

        System.out.print("Library in which the file exists: ");
              library = inputStream.readLine();

        System.out.print("File name: ");
                file = inputStream.readLine();

        System.out.print("Member name (press enter for first member): ");
                member = inputStream.readLine();
        if (member.equals(""))
        {
          member = "*FIRST";
        }

            System.out.println();
      }
                                catch (Exception e)
      {
        System.out.println("Error obtaining user input.");
                e.printStackTrace();
                    System.exit(0);
      }

          // Creare un oggetto AS400 e collegarsi per il servizio di accesso al livello record.
            AS400 system = new AS400(systemName);
                                try
      {
              system.connectService(AS400.RECORDACCESS);
      }
        catch(Exception e)
      {
        System.out.println("Unable to connect for record level access.");
        System.out.println("Check the readme file for special instructions regarding record
                           level access");
                e.printStackTrace();
                    System.exit(0);
      }

            // Creare un oggetto QSYSObjectPathName per ottenere il formato nome percorso integrated file system
            // del file da visualizzare.
      QSYSObjectPathName filePathName = new QSYSObjectPathName(library, file, member, "MBR");

            // Creare un oggetto SequentialFile che rappresenti il file da visualizzare
```

```
            SequentialFile theFile = new SequentialFile(system, filePathName.getPath());

            // Reperire il formato record per il file
        AS400FileRecordDescription recordDescription =
          new AS400FileRecordDescription(system, filePathName.getPath());
                                   try
        {
                  RecordFormat[] format = recordDescription.retrieveRecordFormat();

                                 // Impostare il formato record per il file
                  theFile.setRecordFormat(format[0]);

                  // Aprire il file per la lettura.  Leggere 100 record alla volta se possibile.
          theFile.open(AS400File.READ_ONLY, 100, AS400File.COMMIT_LOCK_LEVEL_NONE);

                  // Visualizzare ogni record nel file
          System.out.println("Displaying file " + library.toUpperCase() + "/"
                             + file.toUpperCase() + "(" + theFile.getMemberName().trim() + "):");

                  Record record = theFile.readNext();
                  while (record != null)
          {
                      System.out.println(record);
                      record = theFile.readNext();
          }
              System.out.println();

                  // Chiudere il file
                  theFile.close();

                  // Scollegarsi dal servizio di accesso al livello record
              system.disconnectService(AS400.RECORDACCESS);
        }
                                   catch (Exception e)
        {
          System.out.println("Error occurred attempting to display the file.");
                  e.printStackTrace();

                                     try
          {
                    // Chiudere il file
                    theFile.close();
          }
                catch(Exception x)
          {
          }

                  // Scollegarsi dal servizio di accesso al livello record
              system.disconnectService(AS400.RECORDACCESS);
                      System.exit(0);
        }

              // Assicurarsi che l'applicazione termini; consultare il readme per i dettagli
                    System.exit(0);
      }
    }
```

Esempio: utilizzo delle classi di accesso al livello record per la lettura dei record da un file

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
///////////////////////////////////////////////////////////////////////////
//
// Esempio di accesso al livello record.  Questo programma utilizza le classi di accesso
// a livello record per leggere i record da un file nel server.
//
```

```
// Sintassi del comando:
//    java RLReadFile system
//
// Questo programma legge i record dal file database di esempio di CA/400
// (QCUSTCDT nella libreria QIWS).  Se si modifica questo esempio per aggiornare i
// record, fare una copia di QCUSTCDT ed aggiornare la copia.
//
// Questo sorgente è un esempio di "Accesso a livello record" di IBM Toolbox per Java.
//
////////////////////////////////////////////////////////////////////////////


import java.io.*;
import java.util.*;
import java.math.*;
import com.ibm.as400.access.*;

public class RLReadFile extends Object
{
      public static void main(String[] parameters)
   {

            String system        = "";

            // Continuare solo se è stato specificato un nome di sistema.

      if (parameters.length >= 1)
      {

                                    try
        {

                        // Supponiamo che il primo parametro sia il nome di sistema.

                        system = parameters[0];

                        // Creare un oggetto AS400 per il server che contiene il file.

                        AS400 as400 = new AS400(system);


                        // Creare una descrizione record per il file.  Il file è QCUSTCDT
           // nella libreria QIWS.

                        ZonedDecimalFieldDescription customerNumber =
                        new ZonedDecimalFieldDescription(new AS400ZonedDecimal(6,0),
                                                         "CUSNUM");
                        CharacterFieldDescription lastName =
                          new CharacterFieldDescription(new AS400Text(8, as400), "LSTNAM");

                        CharacterFieldDescription initials =
                          new CharacterFieldDescription(new AS400Text(3, as400), "INIT");

                        CharacterFieldDescription street =
                          new CharacterFieldDescription(new AS400Text(13, as400), "STREET");

                        CharacterFieldDescription city =
                          new CharacterFieldDescription(new AS400Text(6, as400), "CITY");

                        CharacterFieldDescription state =
                          new CharacterFieldDescription(new AS400Text(2, as400), "STATE");

                        ZonedDecimalFieldDescription zipCode =
                        new ZonedDecimalFieldDescription(new AS400ZonedDecimal(5,0),
                                                         "ZIPCOD");
                        ZonedDecimalFieldDescription creditLimit =
                        new ZonedDecimalFieldDescription(new AS400ZonedDecimal(4,0),
```

```
                                        "CDTLMT");
            ZonedDecimalFieldDescription chargeCode =
            new ZonedDecimalFieldDescription(new AS400ZonedDecimal(1,0),
                                        "CHGCOD");
            ZonedDecimalFieldDescription balanceDue =
            new ZonedDecimalFieldDescription(new AS400ZonedDecimal(6,2),
                                        "BALDUE");
            ZonedDecimalFieldDescription creditDue =
            new ZonedDecimalFieldDescription(new AS400ZonedDecimal(6,2),
                                        "CDTDUE");


// E' necessario specificare il nome formato record per un file DDM.
            // Nel caso del file QCUSTCDT, il relativo formato record viene denominato CUSREC.

RecordFormat qcustcdt = new RecordFormat("CUSREC");

            qcustcdt.addFieldDescription(customerNumber);
            qcustcdt.addFieldDescription(lastName);
            qcustcdt.addFieldDescription(initials);
            qcustcdt.addFieldDescription(street);
            qcustcdt.addFieldDescription(city);
            qcustcdt.addFieldDescription(state);
            qcustcdt.addFieldDescription(zipCode);
            qcustcdt.addFieldDescription(creditLimit);
            qcustcdt.addFieldDescription(chargeCode);
            qcustcdt.addFieldDescription(balanceDue);
            qcustcdt.addFieldDescription(creditDue);


            // Creare l'oggetto file sequenziale che rappresenta il
// file sul server.  Si utilizza un oggetto QSYSObjectPathName
            // per richiamare il nome del file nel formato corretto.

QSYSObjectPathName fileName = new QSYSObjectPathName("QIWS",
                                                     "QCUSTCDT",
                                                     "FILE");

            SequentialFile file = new SequentialFile(as400, fileName.getPath());


            // Consentire all'oggetto file di riconoscere il formato dei record.

            file.setRecordFormat(qcustcdt);


// Aprire il file per un accesso di sola lettura.  Specificare un fattore
            // di blocco di 10 (l'oggetto file richiamerà 10 record quando
            // accede al server per i dati).  Non utilizzare il controllo
// sincronizzazione.

            file.open(SequentialFile.READ_ONLY,
                                10,
                                SequentialFile.COMMIT_LOCK_LEVEL_NONE);


            // Leggere il primo record del file.

            Record data = file.readNext();


            // Eseguire il loop mentre vi sono record nel file (quando non è stata
            // raggiunta l'end-of-file).

            while (data != null)
{
```

```
                        // Visualizzare il record solo se il saldo dovuto è maggiore di
                        // zero.  In tal caso visualizzare il nome del cliente ed
// il saldo dovuto.  Il seguente codice estrae i campi
               // del record per nome campo.  Quando il campo viene richiamato
                        // dal record viene convertito dal formato server nel
                        // formato Java.

               if (((BigDecimal)data.getField("BALDUE")).floatValue() > 0.0)
               {
                  System.out.print((String) data.getField("INIT") + "  ");
                  System.out.print((String) data.getField("LSTNAM") + "  ");
                  System.out.println((BigDecimal) data.getField("BALDUE"));
               }

                               // Leggere il record successivo nel file.

                               data = file.readNext();
            }


                           // Quando non vi sono più record da leggere, scollegarsi dal server.

                           as400.disconnectAllServices();
         }

                                    catch (Exception e)
         {

            // Se una qualsiasi delle operazioni riportate sopra ha dato esito negativo,
            // stampare un messaggio di errore ed emettere l'eccezione.

            System.out.println("Could not read the file");
                    System.out.println(e);
         }
      }


            // Visualizzare il testo di aiuto quando i parametri non sono corretti.

                               else
      {
                  System.out.println("");
                  System.out.println("");
                  System.out.println("");
         System.out.println("Parameters are not correct.  Command syntax is:");
                  System.out.println("");
         System.out.println("   RLReadFile as400");
                  System.out.println("");
                  System.out.println("Where");
                  System.out.println("");
         System.out.println("   as400 = system that contains the file");
                  System.out.println("");
                  System.out.println("For example:");
                  System.out.println("");
         System.out.println("   RLReadFile mySystem");
                  System.out.println("");
                  System.out.println("");
         System.out.println("Note, this program reads data base file QIWS/QCUSTCDT.  ");
                  System.out.println("");
                  System.out.println("");
      }

                  System.exit(0);

   }
}
```

Esempio: utilizzo delle classi di accesso a livello record per leggere record in base alla chiave

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
///////////////////////////////////////////////////////////////////////////
//
// Esempio di accesso al livello record.  Questo programma utilizza le classi di accesso
// a livello record per leggere record per chiave da un file nel server.
// All'utente verrà richiesto il nome del server nel quale effettuare l'esecuzione e
// la libreria nella quale creare il file QCUSTCDTKY.
//
// Sintassi del comando:
//    java RLKeyedFileExample
//
// Questo programma copierà i record dal file database di esempio
// di iSeries Access per Windows (QCUSTCDT nella libreria QIWS) nel file QCUSTCDTKY che ha
// lo stesso formato di QIWS/QCUSTCDT ma ha impostato il campo CUSNUM come chiave
// per il file.
//
// Questo sorgente è un esempio di "Accesso a livello record" di IBM Toolbox per Java.
//
///////////////////////////////////////////////////////////////////////////


import java.io.*;
import java.util.*;
import java.math.*;
import com.ibm.as400.access.*;

public class RLKeyedFileExample
{
     public static void main(String[] parameters)
  {

          // Creato un programma di lettura per richiamare l'immissione dall'utente
          BufferedReader inputStream = new BufferedReader(new InputStreamReader(System.in),1);

        // Dichiarare variabili per contenere il nome di sistema, i nomi libreria, file e membro
    String systemName = "";
    String library = "";

        // Richiamare il nome sistema dall'utente
        System.out.println();
                               try
    {
      System.out.print("System name: ");
            systemName = inputStream.readLine();

      System.out.print("Library in which to create file QCUSTCDTKY: ");
            library = inputStream.readLine();
    }
      catch(Exception e)
    {
      System.out.println("Error obtaining user input.");
              e.printStackTrace();
                  System.exit(0);
    }

        // Creare un oggetto AS400 e collegarsi per il servizio di accesso al livello record.
          AS400 system = new AS400(systemName);
                               try
    {
            system.connectService(AS400.RECORDACCESS);
    }
      catch(Exception e)
    {
```

```
  System.out.println("Unable to connect for record level access.");
  System.out.println("Check the readme file for special instructions regarding record
                    level access");
          e.printStackTrace();
              System.exit(0);
}

    RecordFormat qcustcdtFormat = null;
                           try
{
  // Creare l'oggetto RecordFormat per la creazione del file.  Il formato record per il nuovo
   // file sarà uguale al formato record per il file QIWS/QCUSTCDT.  Tuttavia si renderà
   // il campo CUSNUM un campo chiave.
  AS400FileRecordDescription recordDescription =
    new AS400FileRecordDescription(system, "/QSYS.LIB/QIWS.LIB/QCUSTCDT.FILE");

        // Vi è un solo formato record per il file, quindi prendere il primo (ed unico) elemento
        // della schiera RecordFormat restituito come RecordFormat per il file.
  System.out.println("Retrieving record format of QIWS/QCUSTCDT...");
        qcustcdtFormat = recordDescription.retrieveRecordFormat()[0];
        // Indicare CUSNUM come campo chiave
  qcustcdtFormat.addKeyFieldDescription("CUSNUM");
}
  catch(Exception e)
{
  System.out.println("Unable to retrieve record format from QIWS/QCUSTCDT");
          e.printStackTrace();
              System.exit(0);
}

    // Creare l'oggetto file con chiave che rappresenta il
    // file che verrà creato sul server.  Si utilizza un oggetto QSYSObectPathName
     // per richiamare il nome del file nel formato corretto.
    QSYSObjectPathName fileName = new QSYSObjectPathName(library,
                                                         "QCUSTCDTKY",
                                                         "*FILE",
                                                         "MBR");
    KeyedFile file = new KeyedFile(system, fileName.getPath());

                           try
{
  System.out.println("Creating file " + library + "/QCUSTCDTKY...");
        // Creare il file utilizzando l'oggetto qcustcdtFormat
  file.create(qcustcdtFormat, "Keyed QCUSTCDT file");

        // Popolare il file con i record contenuti in QIWS/QCUSTCDT
        copyRecords(system, library);

        // Aprire il file per un accesso di sola lettura.  Poiché si accederà
        // casualmente al file, specificare un fattore di blocco di 1. Le
        // il parametro livello blocco di sincronizzazione verrà ignorato poiché il controllo
        // sincronizzazione non è stato ancora avviato.
        file.open(AS400File.READ_ONLY,
                           1,
                           AS400File.COMMIT_LOCK_LEVEL_NONE);

        // Supponiamo di voler visualizzare le informazioni per i clienti
        // 192837, 392859 e 938472
        // Il campo CUSNUM è un campo decimale a zonatura di lunghezza 6 senza
        // posizioni decimali.  Pertanto, il valore del campo chiave è
         // rappresentato da un BigDecimal.
        BigDecimal[] keyValues = {new BigDecimal(192837), new BigDecimal(392859), new BigDecimal(938472)};
  // Creare la chiave di lettura dei record.  La chiave per un KeyedFile
  // è specificata con un Object[]
        Object[] key = new Object[1];

        Record data = null;
```

```
          for (int i = 0; i < keyValues.length; i++)
    {
              // Impostare la chiave di lettura
              key[0] = keyValues[i];

              // Leggere il record per keyValues[i] del numero cliente
              data = file.read(key);
              if (data != null)
      {
                  // Visualizzare il record solo se il saldo dovuto è maggiore di
                  // zero.  In tal caso visualizzare il nome del cliente ed
                   // il saldo dovuto.  Il seguente codice estrae i campi
                  // del record per nome campo.  Quando il campo viene richiamato
                  // dal record lo si converte dal formato server nel
                  // formato Java.
        if (((BigDecimal)data.getField("BALDUE")).floatValue() > 0.0)
        {
          System.out.print((String) data.getField("INIT") + "  ");
          System.out.print((String) data.getField("LSTNAM") + "  ");
          System.out.println((BigDecimal) data.getField("BALDUE"));
        }
      }
    }

          //  Terminato il lavoro sul file
            file.close();

          // Eliminare il file dal sistema dell'utente
            file.delete();
  }
    catch(Exception e)
  {
    System.out.println("Unable to create/read from QTEMP/QCUSTCDT");
            e.printStackTrace();
                              try
    {
              file.close();
            // Eliminare il file dal sistema dell'utente
              file.delete();
    }
          catch(Exception x)
    {
    }
  }

      // Fine delle operazioni relative all'accesso a livello record; scollegarsi
      // dal server di accesso a livello record.
      system.disconnectService(AS400.RECORDACCESS);
              System.exit(0);
}

  public static void copyRecords(AS400 system, String library)
{
      // Utilizzare la classe CommandCall per eseguire il comando CPYF per copiare i record
      // da QIWS/QCUSTCDT a QTEMP/QCUSTCDT
  CommandCall c = new CommandCall(system, "CPYF FROMFILE(QIWS/QCUSTCDT) TOFILE("
                                  + library + "/QCUSTCDTKY) MBROPT(*REPLACE)");
                               try
  {
    System.out.println("Copying records from QIWS/QCUSTCDT to "
                       + library + "/QCUSTCDTKY...");
          c.run();
          AS400Message[] msgs = c.getMessageList();
    if (!msgs[0].getID().equals("CPC2955"))
    {
      System.out.println("Unable to populate " + library + "/QCUSTCDTKY");
              for (int i = 0; i < msgs.length; i++)
```

```
        {
                    System.out.println(msgs[i]);
        }
                    System.exit(0);
      }
    }
      catch(Exception e)
    {
      System.out.println("Unable to populate " + library + "/QCUSTCDTKY");
                  System.exit(0);
    }
  }
}
```

Esempio: utilizzo di UserList per elencare tutti gli utenti in un determinato gruppo

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
///////////////////////////////////////////////////////////////////////////
//
// Esempio di User list.  Questo programma elenca tutti gli utenti in un determinato
// gruppo.
//
// Sintassi del comando:
//    UserListExample system group
//
// Questo sorgente è un esempio di "UserList" di IBM Toolbox per Java.
//
///////////////////////////////////////////////////////////////////////////

import com.ibm.as400.access.*;
import com.ibm.as400.vaccess.*;
import java.util.Enumeration;

public class UserListExample
{


        public static void main (String[] args)
    {
        // Se non sono stati specificati un sistema ed un gruppo, visualizzare
        // il testo di aiuto ed uscire.
                if (args.length != 2)
        {
            System.out.println("Usage:  UserListExample system group");
                        return;
        }

                                    try
        {
                    // Creare un oggetto AS400.
Il nome di sistema è stato passato
                        // come primo argomento della riga comandi.
                        AS400 system = new AS400 (args[0]);

                        // Il nome gruppo è stato passato come secondo argomento della riga
                        // comandi.
                        String groupName = args[1];

                        // Creare l'oggetto elenco utenti.
                        UserList userList = new UserList (system);

                        // Richiamare un elenco degli utenti in un dato gruppo.
                        userList.setUserInfo (UserList.MEMBER);
                        userList.setGroupInfo (groupName);
                        Enumeration enum = userList.getUsers ();
```

```
                        // Ripetere nell elenco e stampare i
                        // nomi e le descrizioni degli utenti.
                        while (enum.hasMoreElements ())
            {
                                User u = (User) enum.nextElement ();
                System.out.println ("User name:   " + u.getName ());
                System.out.println ("Description: " + u.getDescription ());
                         System.out.println("");
            }

        }
                                    catch (Exception e)
        {
                       System.out.println ("Error: " + e.getMessage ());
        }

              System.exit(0);
    }


}
```

Esempi: JavaBean

Questa sezione elenca gli esempi di codice che sono forniti in tutte le informazioni sul bean di IBM Toolbox per Java.

- Esempio: utilizzo dei listener per stampare un commento quando ci si collega e scollega dal sistema e si eseguono comandi
- Esempio: utilizzo delle applet e di IBM VisualAge per Java per creare pulsanti che eseguono comandi

Il seguente esonero di responsabilità si applica a tutti gli esempi IBM Toolbox per Java:

> **Esonero di responsabilità per gli esempi di codice**
>
> L'IBM fornisce una licenza non esclusiva per utilizzare tutti gli esempi del codice di programmazione da cui creare funzioni simili personalizzate, in base a richieste specifiche.
>
> Questo codice di esempio è fornito dall'IBM con la sola funzione illustrativa. Questi esempi non sono stati interamente testati in tutte le condizioni. IBM, perciò, non fornisce nessun tipo di garanzia o affidabilità implicita, rispetto alla funzionalità o alle funzioni di questi programmi.
>
> Tutti i programmi qui contenuti vengono forniti all'utente ″COSI' COME SONO″ senza garanzie di alcun tipo. Le garanzie implicite di non contraffazione, commerciabilità e adeguatezza a scopi specifici sono espressamente vietate.

Esempio: codice bean IBM Toolbox per Java

Il seguente esempio crea un oggetto AS400 e un oggetto CommandCall e quindi registra i listener sugli oggetti. I listener sugli oggetti stampano un commento quando il server si collega o si scollega e quando l'oggetto CommandCall completa l'esecuzione di un comando.

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per informazioni legali importanti.

```
///////////////////////////////////////////////////////////////////////////
//
// Esempio di bean. Questo programma utilizza il supporto JavaBeans nelle
// classi di IBM Toolbox per Java.
//
// Sintassi del comando:
//    BeanExample
//
```

```
////////////////////////////////////////////////////////////////////////////

import com.ibm.as400.access.AS400;
import com.ibm.as400.access.CommandCall;
import com.ibm.as400.access.ConnectionListener;
import com.ibm.as400.access.ConnectionEvent;
import com.ibm.as400.access.ActionCompletedListener;
import com.ibm.as400.access.ActionCompletedEvent;

class BeanExample
{
        AS400       as400_  = new AS400();
        CommandCall cmd_    = new CommandCall( as400_ );

        BeanExample()
    {
                // Ogni volta che il sistema è collegato o scollegato stampare un
        // commento. Effettuare questa operazione aggiungendo un listener all'oggetto AS400.
                // Quando un sistema è collegato o scollegato, l'oggetto AS400
                // chiamerà questo codice.

                as400_.addConnectionListener
                (new ConnectionListener()
          {
                          public void connected(ConnectionEvent event)
             {
                 System.out.println( "System connected." );
             }
                          public void disconnected(ConnectionEvent event)
             {
                 System.out.println( "System disconnected." );
             }
          }
        );

        // Ogni volta che il sistema completa l'esecuzione stampare un commento. Effettuare questa operazione
        // aggiungendo un listener all'oggetto commandCall. L'oggetto The commandCall
                // richiamerà questo codice quando esegue un comando.

                cmd_.addActionCompletedListener(
                        new ActionCompletedListener()
             {
                                public void actionCompleted (ActionCompletedEvent event)
                {
                    System.out.println( "Command completed." );
                }
             }
        );
    }

        void runCommand()
    {
                                    try
        {
            // Eseguire un comando. I listener stamperanno i commenti quando il
                        // sistema è collegato e quando il comando ha completato
                        // l'esecuzione.
                        cmd_.run( "TESTCMD PARMS" );
        }
                            catch (Exception ex)
        {
                        System.out.println( ex );
        }
    }

      public static void main(String[] parameters)
    {
```

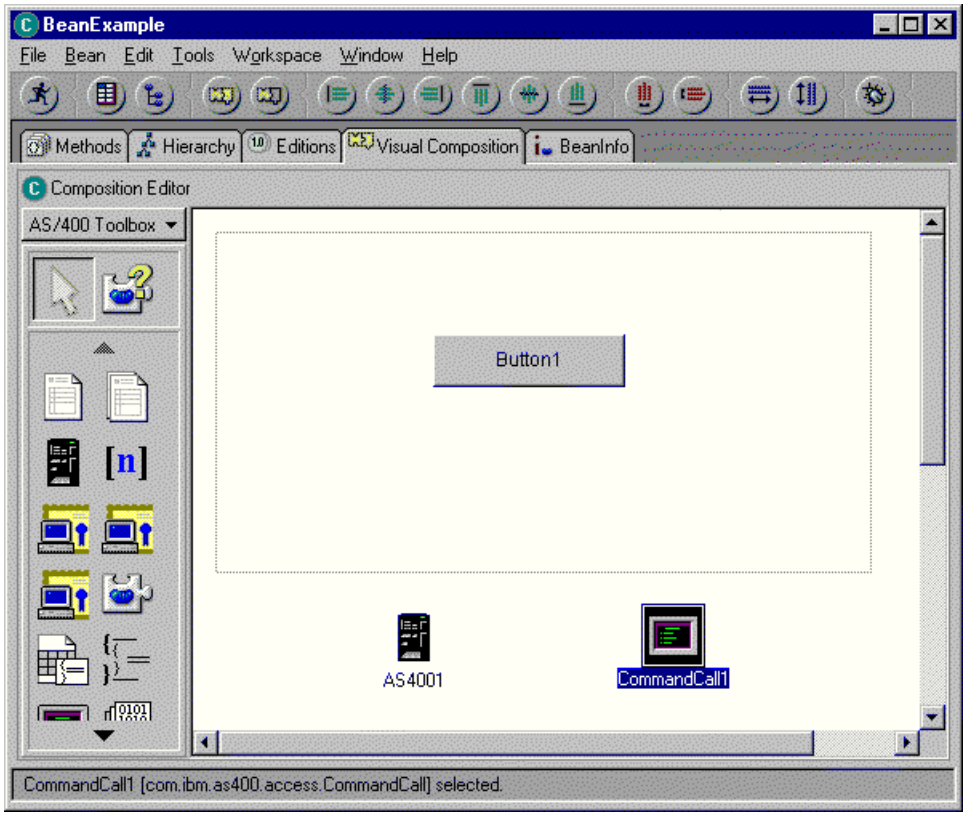
```
            BeanExample be = new BeanExample();

            be.runCommand();

        System.exit(0);
    }
}
```

Esempio: creazione di bean con un programma di creazione del bean visuale

Questo esempio utilizza IBM VisualAge per Java Enterprise Edizione V2.0 Composition Editor, ma altri programmi di creazione di bean visuali sono simili. In questo esempio viene creato un'applet per un pulsante che, quando viene premuto, esegue un comando sul server iSeries.

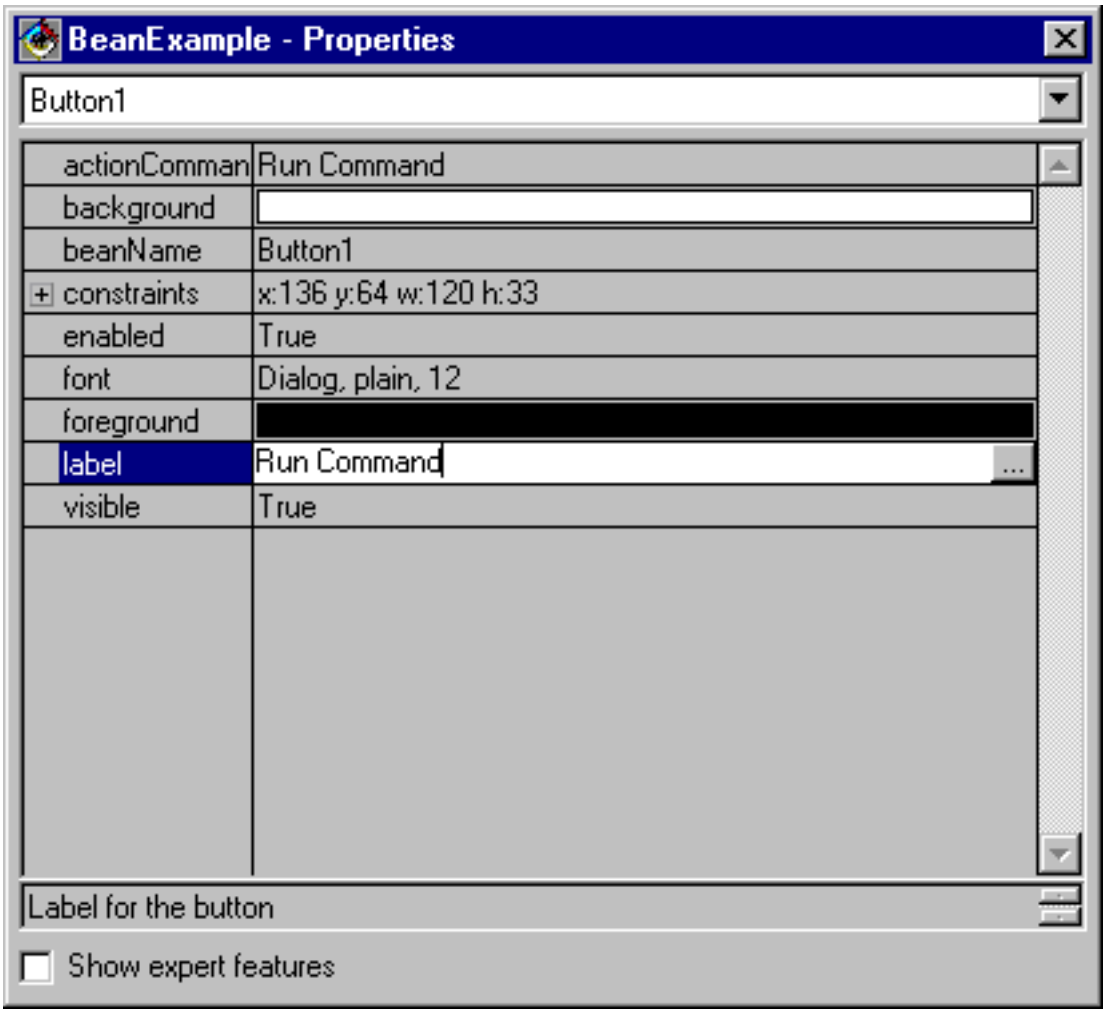
- Trascinare e rilasciare un pulsante sull'applet. (Il pulsante può essere trovato nel programma di creazione bean sulla parte sinistra del separatore della Composizione visuale nella Figura 1.)
- Rilasciare un bean CommandCall e un bean AS400 al di fuori dell'applet. (I bean si trovano nel programma di creazione di bean nella parte sinistra del separatore Composizioni visuale nella Figura 1.)

Figura 1: finestra Editor di Composizione visuale VisualAge - gui.BeanExample



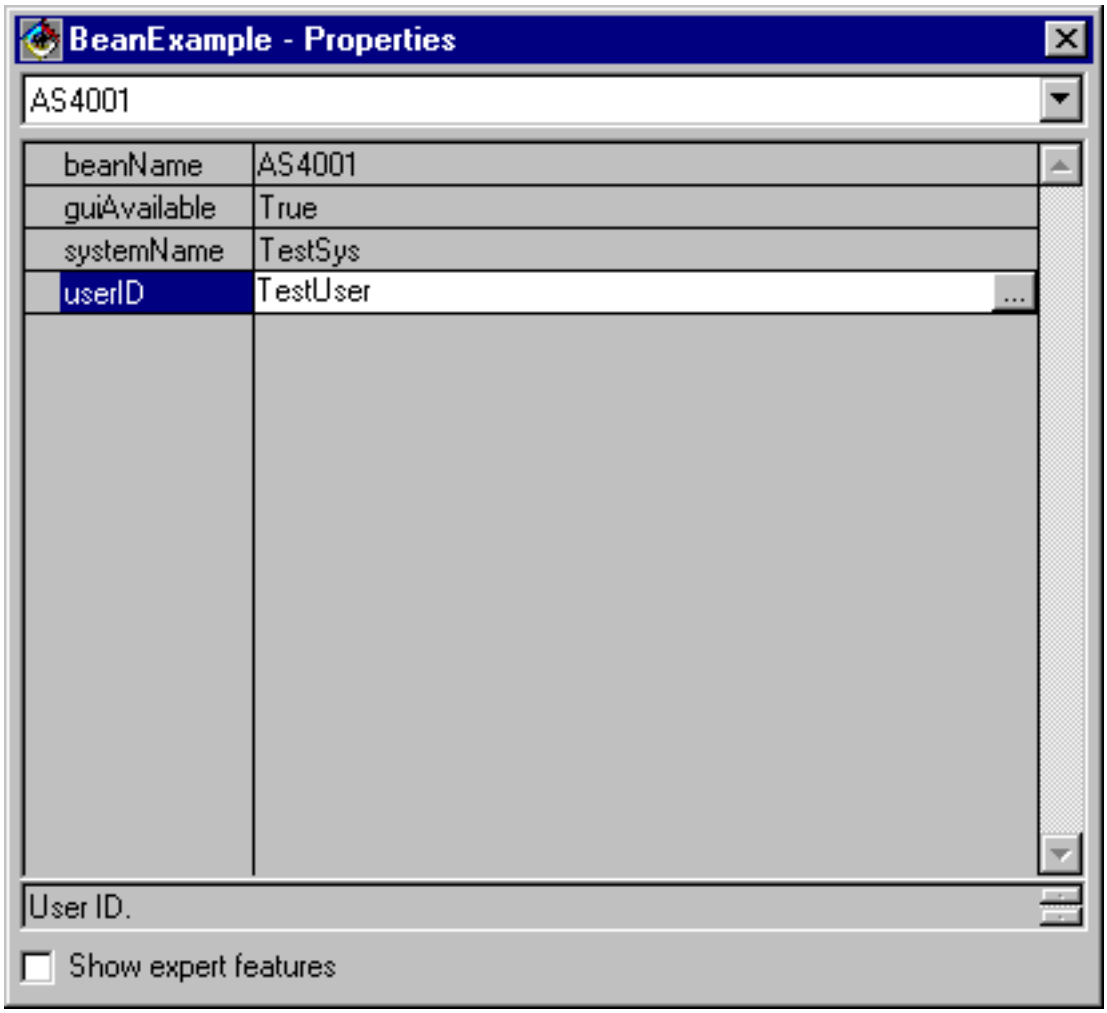
- Modificare le proprietà del bean. (Per effettuare la modifica, selezionare il bean e quindi fare clic con il tastino destro del mouse per visualizzare una finestra, che dispone dell'opzione Proprietà).
  - Modificare l'etichetta del pulsante in **Esegui comando**, come mostrato nella Figura 2.

    Figura 2: modifica dell'etichetta del pulsante in Esegui comando



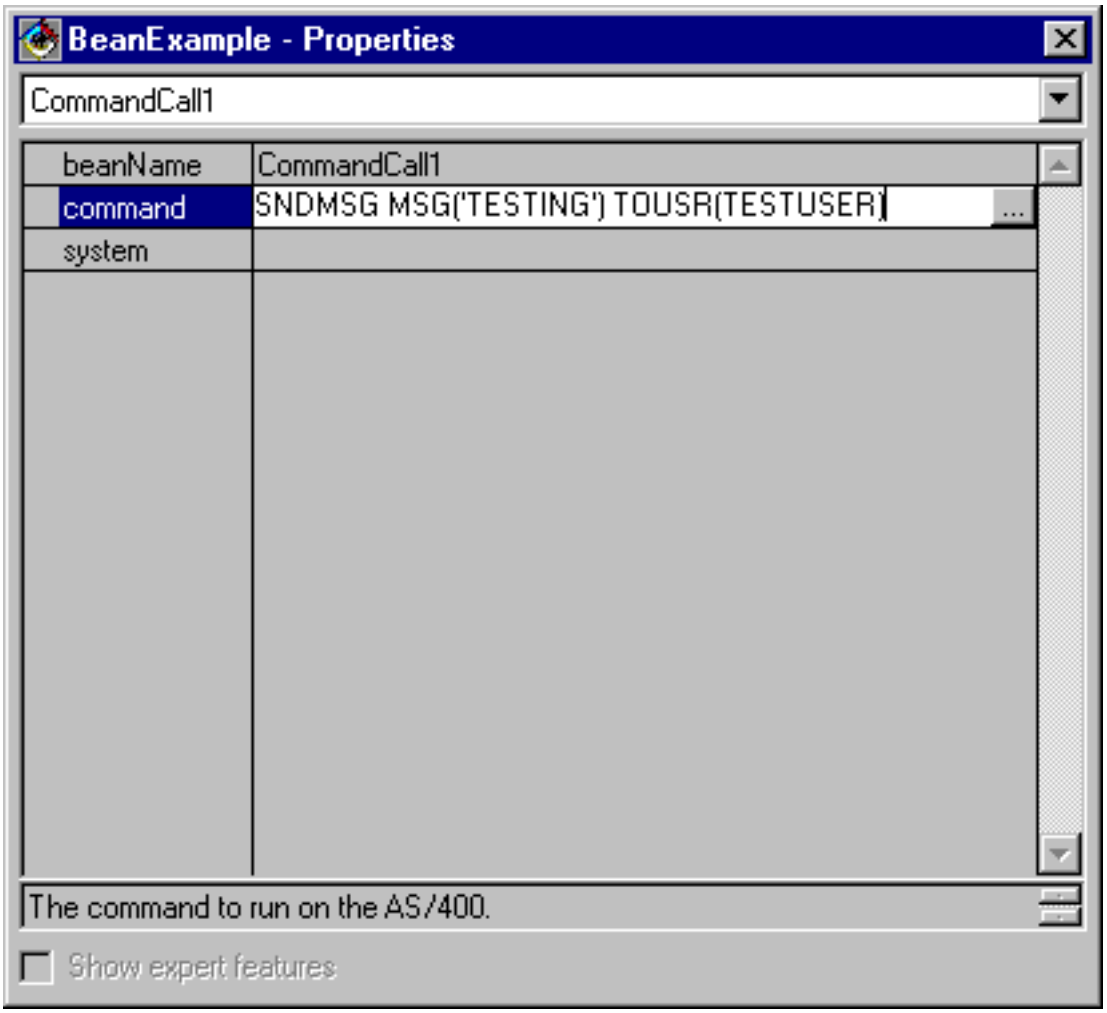
– Modificare il nome del sistema del bean AS400 in **TestSys**
– Modificare l'ID utente del bean AS400 in **TestUser**, come mostrato nella Figura 3.

Figura 3: modifica del nome dell'ID utente in TestUser



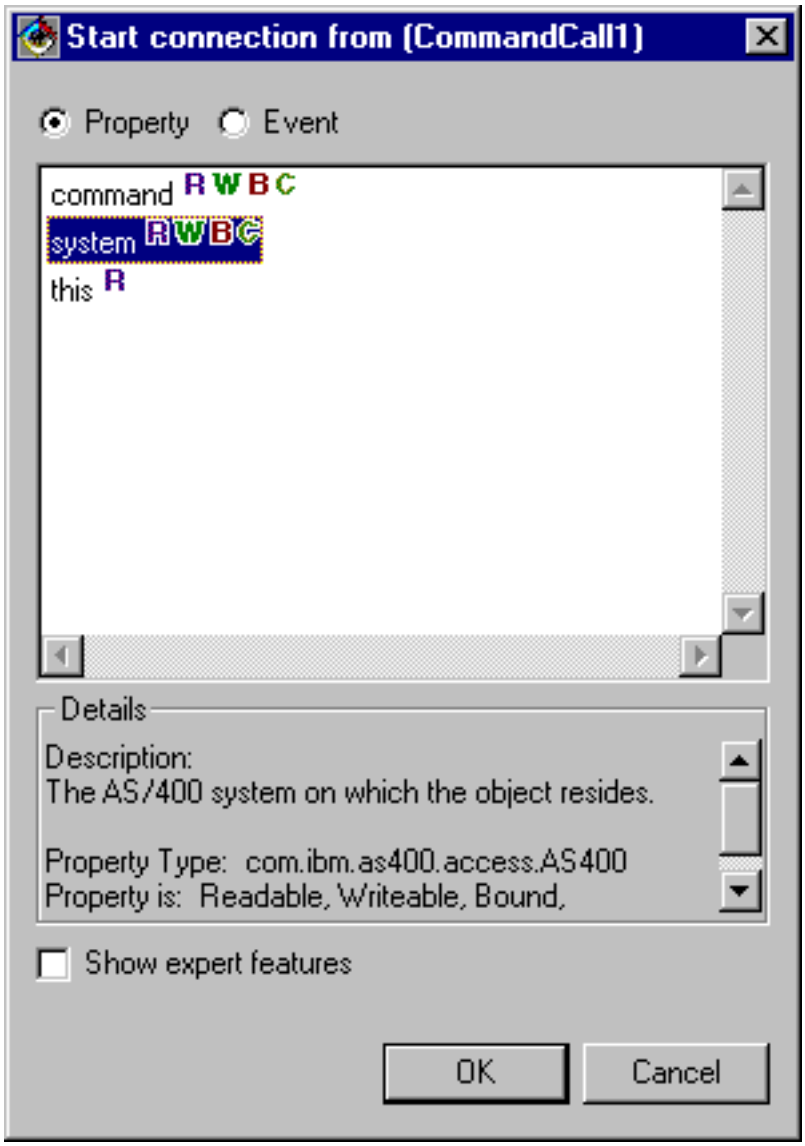
– Modificare il comando del bean CommandCall in **SNDMSG MSG('Testing') TOUSR('TESTUSER')**, come mostrato nella Figura 4.

Figura 4: modifica del comando del bean CommandCall



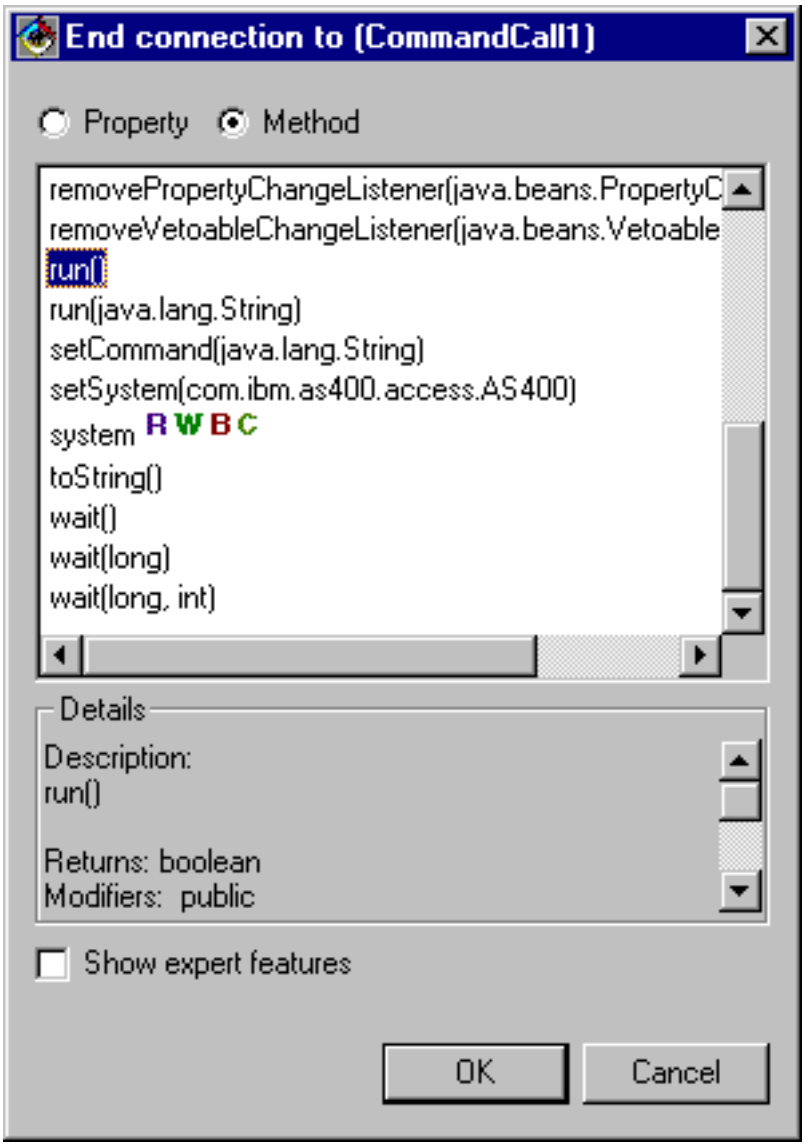
- Collegare il bean AS400 al bean CommandCall. Il metodo che si utilizza per compiere questa operazione varia a seconda dei programmi di creazione del bean. Per questo esempio, effettuare le seguenti operazioni:
  - Selezionare il bean CommandCall, quindi fare clic con il tastino destro del mouse
  - Selezionare **Collegamento**
  - Selezionare **Dispositivi di collegamento**
  - Selezionare **sistema** dall'elenco di dispositivi come mostrato nella Figura 5.
  - Selezionare il bean AS400
  - Selezionare **questo bean** dal menu a comparsa visualizzato sul bean AS400

  Figura 5: collegamento del bean AS400 al bean CommandCall



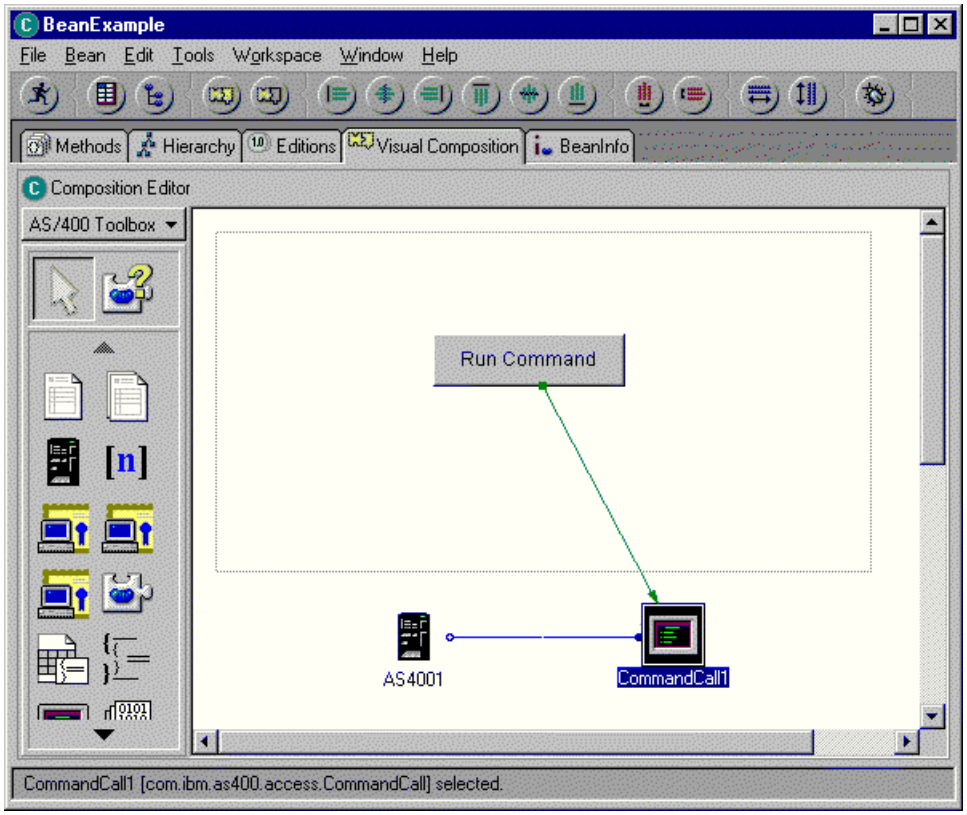
- Collegare il pulsante al bean CommandCall.
  - Selezionare il bean Pulsante, quindi fare clic con il tastino destro del mouse
  - Selezionare **Collegamento**
  - Selezionare **actionPerformed**
  - Selezionare il bean CommandCall
  - Selezionare **Dispositivi di collegamento** dal menu a comparsa visualizzato
  - Selezionare **run()** dall'elenco di metodi come mostrato nella Figura 6.

    Figura 6: collegamento di un metodo al pulsante



Terminata l'operazione, la finestra Editor di Composizione visuale di VisualAge dovrebbe apparire come nella Figura 7.

Figura 7: finestra Editor di composizione visuale di VisualAge - Esempio di bean terminato



Esempi: classi Commtrace

Questa pagina si collega all'esempio di codice fornito nella documentazione delle classi CommTrace di IBM Toolbox per Java.

- "Esempio: utilizzo delle classi commtrace" a pagina 188

Il seguente esonero di responsabilità si applica a tutti gli esempi IBM Toolbox per Java:

> **Esonero di responsabilità per gli esempi di codice**
>
> L'IBM fornisce una licenza non esclusiva per utilizzare tutti gli esempi del codice di programmazione da cui creare funzioni simili personalizzate, in base a richieste specifiche.
>
> Questo codice di esempio è fornito dall'IBM con la sola funzione illustrativa. Questi esempi non sono stati interamente testati in tutte le condizioni. IBM, perciò, non fornisce nessun tipo di garanzia o affidabilità implicita, rispetto alla funzionalità o alle funzioni di questi programmi.
>
> Tutti i programmi qui contenuti vengono forniti all'utente "COSI' COME SONO" senza garanzie di alcun tipo. Le garanzie implicite di non contraffazione, commerciabilità e adeguatezza a scopi specifici sono espressamente vietate.

Esempi di Graphical Toolbox

Vengono forniti degli esempi per mostrare le modalità di implementazione degli strumenti in Graphical Toolbox per i propri programmi UI.

- Creazione e visualizzazione di un pannello: tale esempio mostra le funzioni di base e le operazioni dell'ambiente Graphical Toolbox nel complesso

- Creazione e visualizzazione di un pannello: mostra le modalità di creazione e visualizzazione di un pannello quando il pannello ed il file delle proprietà si trovano nello stesso indirizzario
- Creazione di una finestra di dialogo completamente funzionante: mostra le modalità di creazione di una finestra di dialogo completamente funzionante (dopo aver implementato i DataBeans che forniscono i dati a un pannello e identificato gli attributi in PDML)
- Dimensionamento di un pannello utilizzando il gestore pannelli dinamico: mostra in che modo il gestore pannelli dinamico regola dinamicamente il pannello in fase di esecuzione
- Casella combinata modificabile: mostra come codificare un bean di dati per una casella combinata modificabile

I seguenti esempio mostrano in che modo il GUI Builder può essere utile per creare vari elementi GUI:
- Pannelli: mostra la creazione di un pannello di esempio e del codice bean di dati che viene eseguito sul pannello
- Pannelli sovrapposti: mostra la modalità di creazione di un pannello sovrapposto e l'aspetto finale di quest'ultimo
- Fogli delle proprietà: mostra la modalità di creazione di un foglio delle proprietà e l'aspetto finale di quest'ultimo
- Pannelli suddivisi: mostra la modalità di creazione di un pannello suddiviso e l'aspetto finale di quest'ultimo
- Pannelli con separatori: mostra la modalità di creazione di un pannello con separatori e l'aspetto di quest'ultimo
- Wizard: mostra la modalità di creazione di un wizard e l'aspetto finale di quest'ultimo
- Barre degli strumenti: mostra la modalità di creazione di una barra degli strumenti e l'aspetto finale di quest'ultima
- barre dei menu: mostra la modalità di creazione di una barra dei menu e l'aspetto finale di quest'ultima
- Aiuto: mostra la modalità di creazione di un Documento di aiuto e mostra come suddividere il Documento di aiuto in pagine degli argomenti. Consultare anche Modifica dei documenti di aiuto creati dal GUI Builder
- Esempio: mostra l'aspetto di un intero programma PDML, inclusi i pannelli, un foglio delle proprietà, un wizard, il pulsante seleziona/deseleziona e le opzioni di menu.

Il seguente esonero di responsabilità si applica a tutti gli esempi IBM Toolbox per Java:

> **Esonero di responsabilità per gli esempi di codice**
>
> L'IBM fornisce una licenza non esclusiva per utilizzare tutti gli esempi del codice di programmazione da cui creare funzioni simili personalizzate, in base a richieste specifiche.
>
> Questo codice di esempio è fornito dall'IBM con la sola funzione illustrativa. Questi esempi non sono stati interamente testati in tutte le condizioni. IBM, perciò, non fornisce nessun tipo di garanzia o affidabilità implicita, rispetto alla funzionalità o alle funzioni di questi programmi.
>
> Tutti i programmi qui contenuti vengono forniti all'utente "COSI' COME SONO" senza garanzie di alcun tipo. Le garanzie implicite di non contraffazione, commerciabilità e adeguatezza a scopi specifici sono espressamente vietate.

Esempio: creazione di un pannello con il GUI Builder

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per informazioni legali importanti.

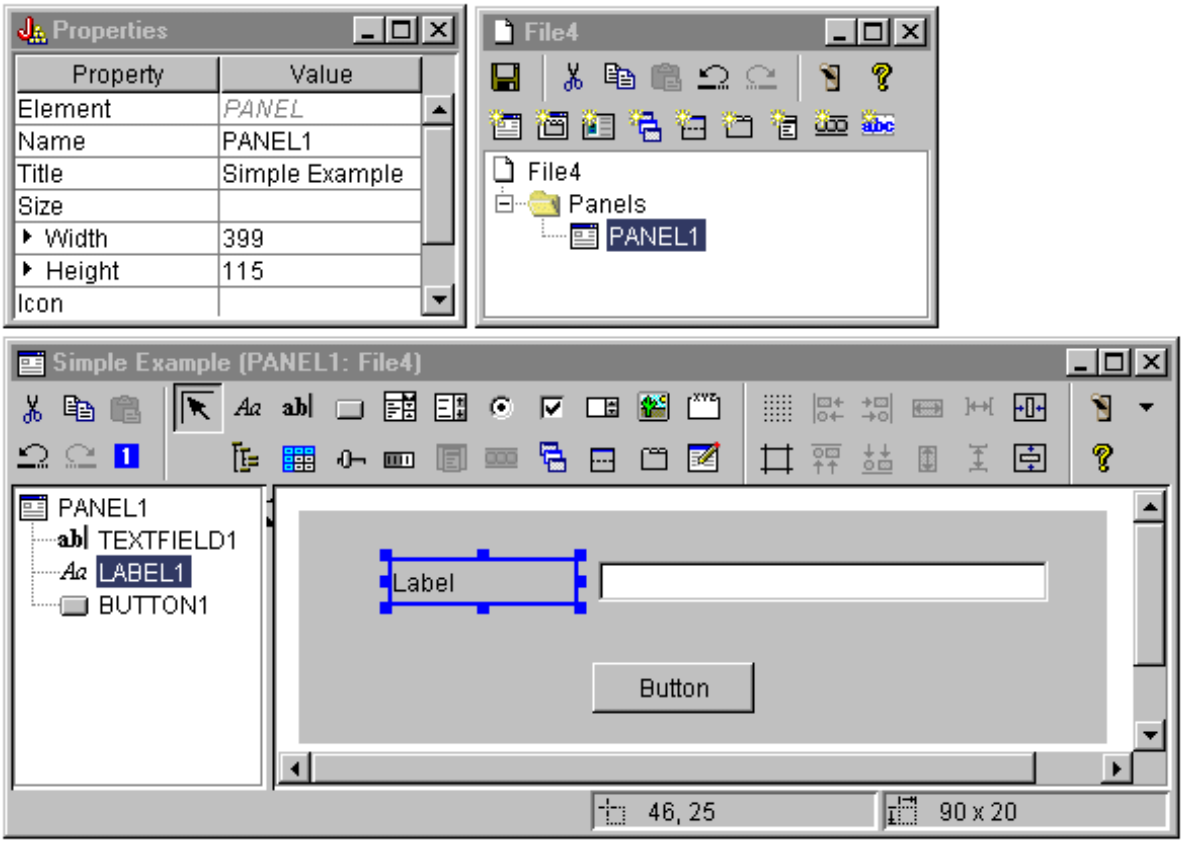
Questo esempio mostra il modo in cui utilizzare il Graphical Toolbox per costruire un pannello semplice. E' una panoramica che illustra le funzioni di base e le operazioni dell'ambiente Graphical Toolbox. Dopo

aver mostrato come creare un pannello, l'esempio continua mostrando come creare una piccola applicazione Java che visualizza il pannello. In questo esempio, l'utente immette i dati in un campo di testo e fa clic sul pulsante **Chiudi**.L'applicazione, quindi, rimanda i dati alla console Java.

Costruzione del pannello

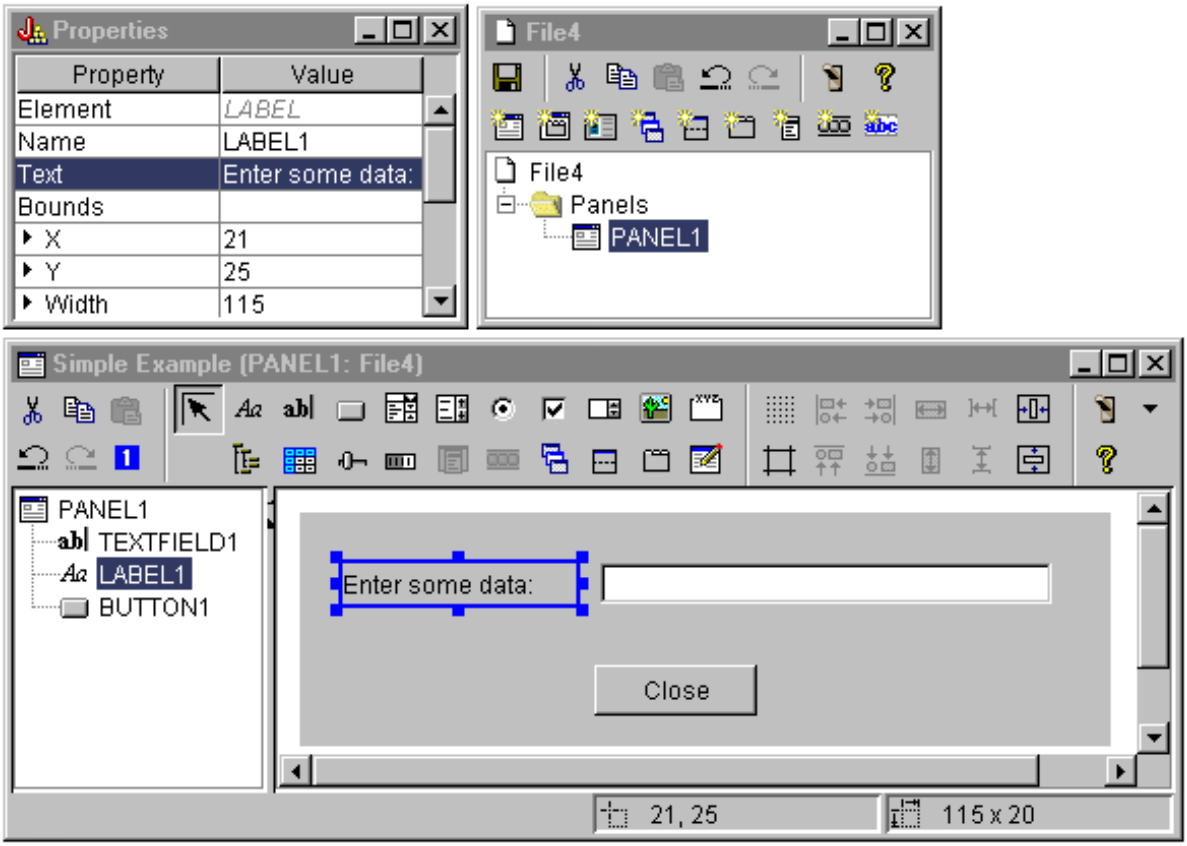
Quando si avvia il GUI Builder, vengono visualizzate le finestre Proprietà e GUI Builder. Creare un nuovo file denominato "MyGUI.pdml". Per questo esempio, inserire un nuovo pannello. Fare clic sull'icona "Inserisci pannello" nella finestra Builder del file. Il suo nome è "PANEL1". Modificare il titolo modificando le informazioni nella finestra Proprietà; immettere "Esempio semplice" nel campo"Titolo". Rimuovere i tre pulsanti predefiniti selezionandoli con il mouse e premendo "Cancella". Utilizzando i pulsanti nella finestra Builder del pannello, aggiungere i tre elementi mostrati nella Figura 1: una etichetta, un campo di testo e un pulsante.

Figura 1: finestre GUI Builder: inizio costruzione di pannello



Selezionando l'etichetta, è possibile modificare il testo nella finestra Proprietà. In questo esempio, lo stesso è stato fatto per il pulsante, modificando il testo in "Chiudi".

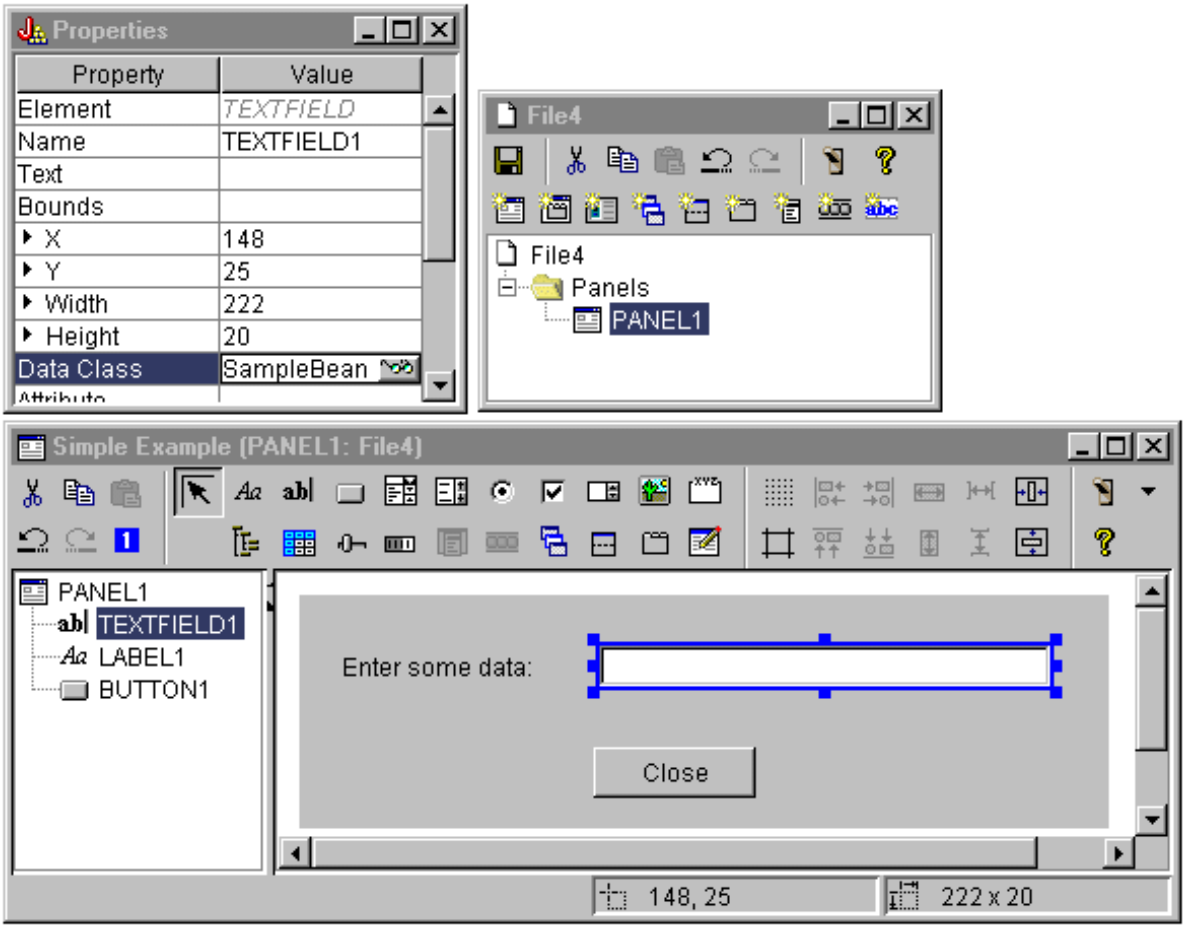
Figura 2: finestre GUI Builder: modifica del testo nella finestra Proprietà



Campo di testo

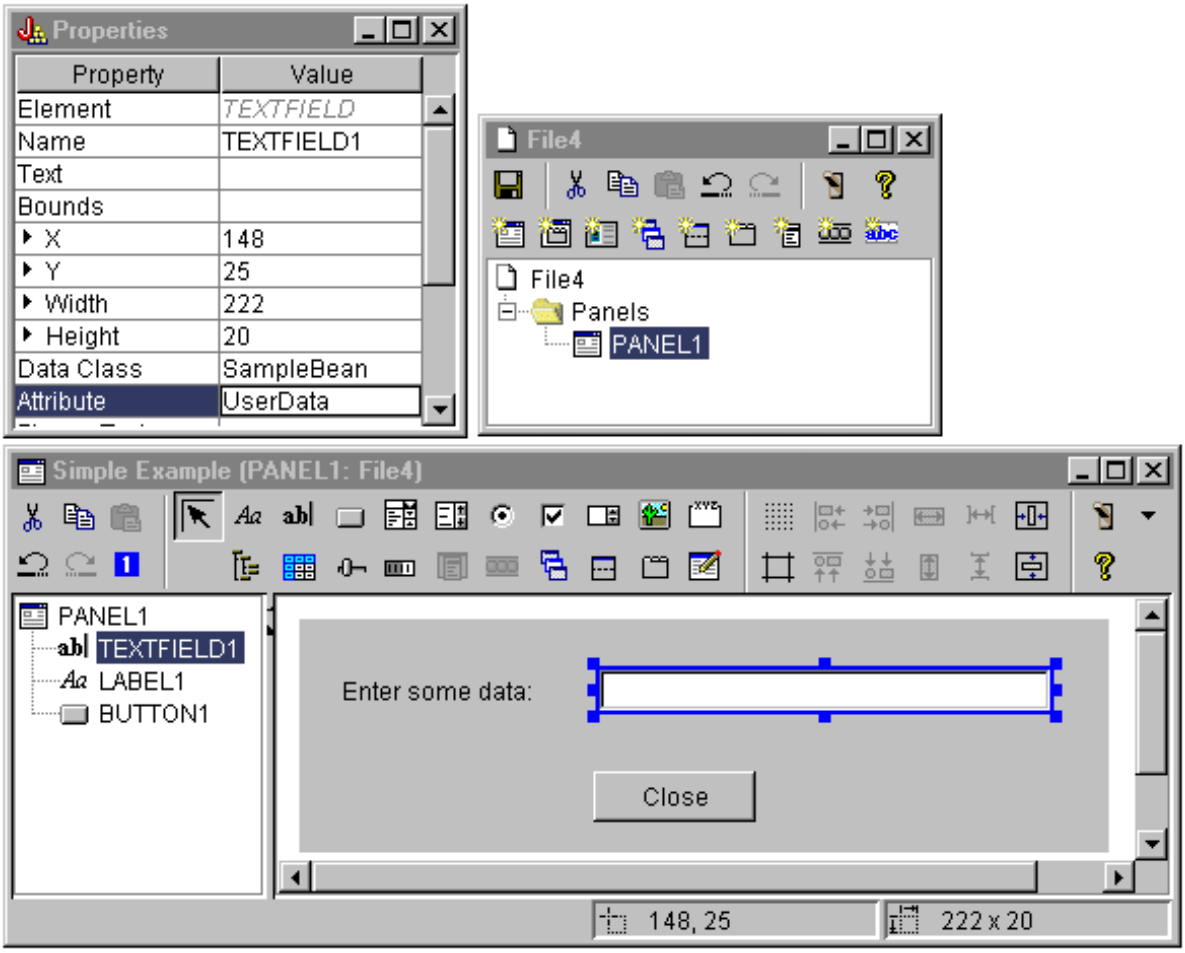
Il campo di testo contiene dati, quindi, è possibile impostare numerose proprietà che consentono al GUI Builder di eseguire del lavoro aggiuntivo. Per questo esempio, si imposta la proprietà Classe dati sul nome di una classe bean denominata **SampleBean**. Questo databean fornirà i dati per questo campo dati.

Figura 3: finestre GUI Builder: impostazione della proprietà Classe dati



Impostare la proprietà Attributo sul nome della proprietà del bean che conterrà i dati. In questo caso, il nome è **UserData**.

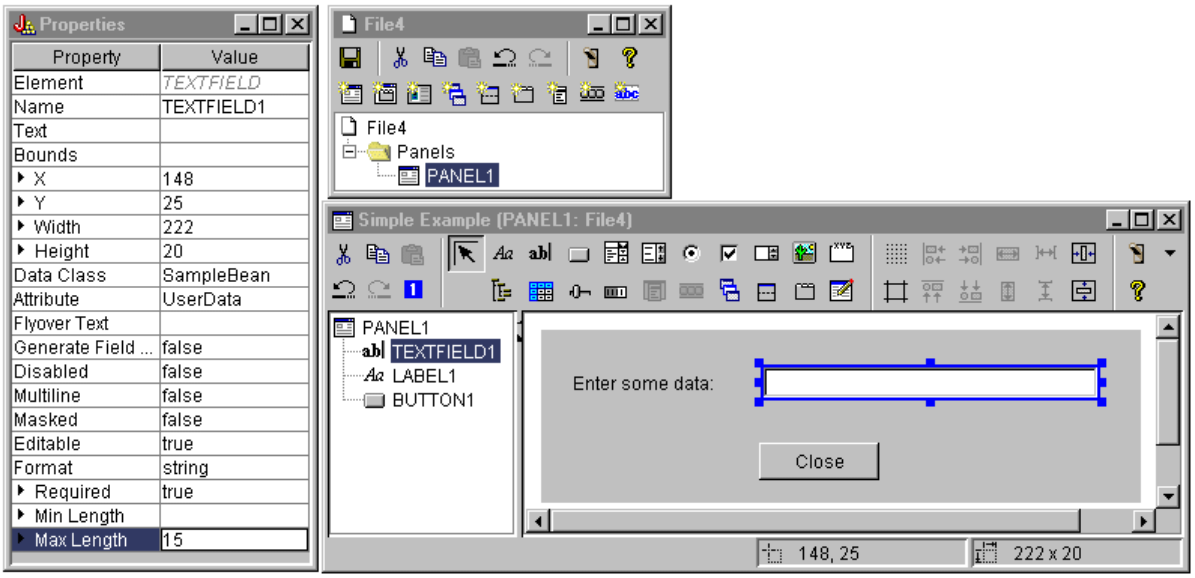
Figura 4: finestre GUI Builder: impostazione della proprietà Attributo



Seguendo le istruzioni precedenti si collega la proprietà **UserData** a questo campo di testo. In fase di esecuzione, il Graphical Toolbox ottiene il valore iniziale di tale campo richiamando **SampleBean.getUserData**. Il valore modificato viene quindi restituito all'applicazione quando il pannello si chiude richiamando **SampleBean.setUserData**.

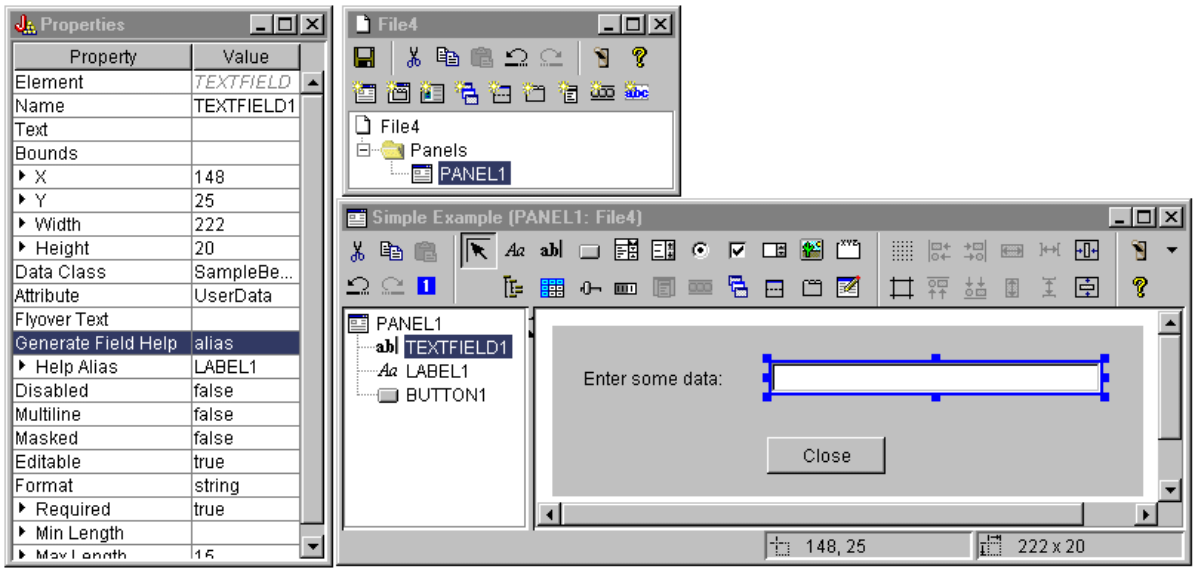
Specificare che all'utente viene richiesto di fornire alcuni dati e che i dati devono essere contenuti in una stringa con una lunghezza massima di 15 caratteri.

Figura 5: finestre GUI Builder: impostazione della lunghezza massima del campo di testo



Indicare che l'aiuto sensibile al contesto per il campo di testo sarà l'argomento di aiuto associato all'etichetta "Immettere alcuni dati".

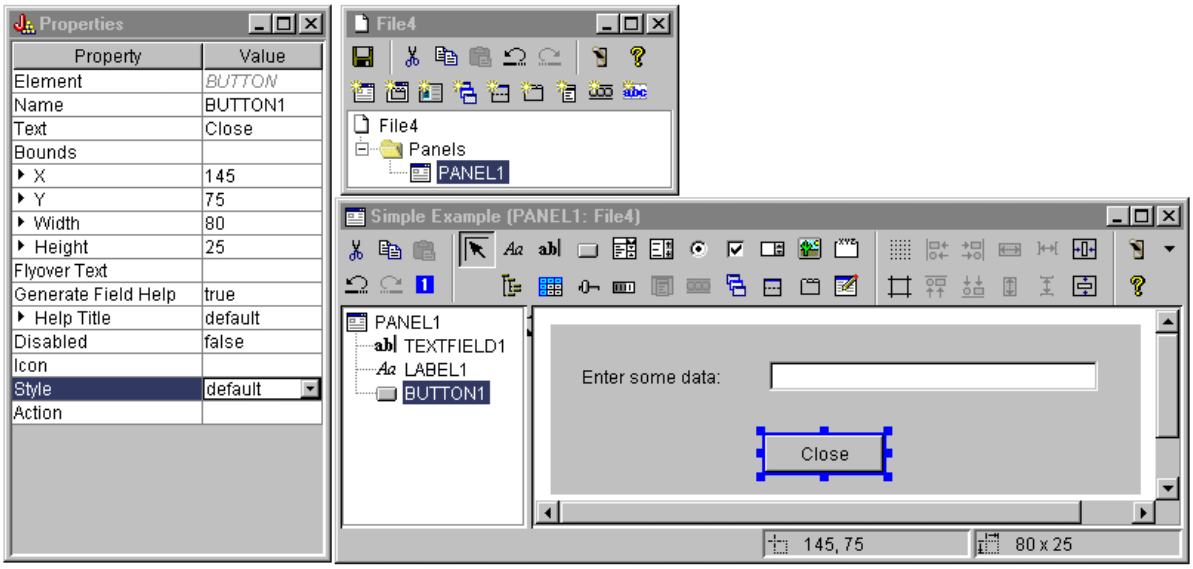
Figura 6: finestre GUI Builder: impostazione dell'aiuto sensibile al contesto per il campo di testo



Pulsante

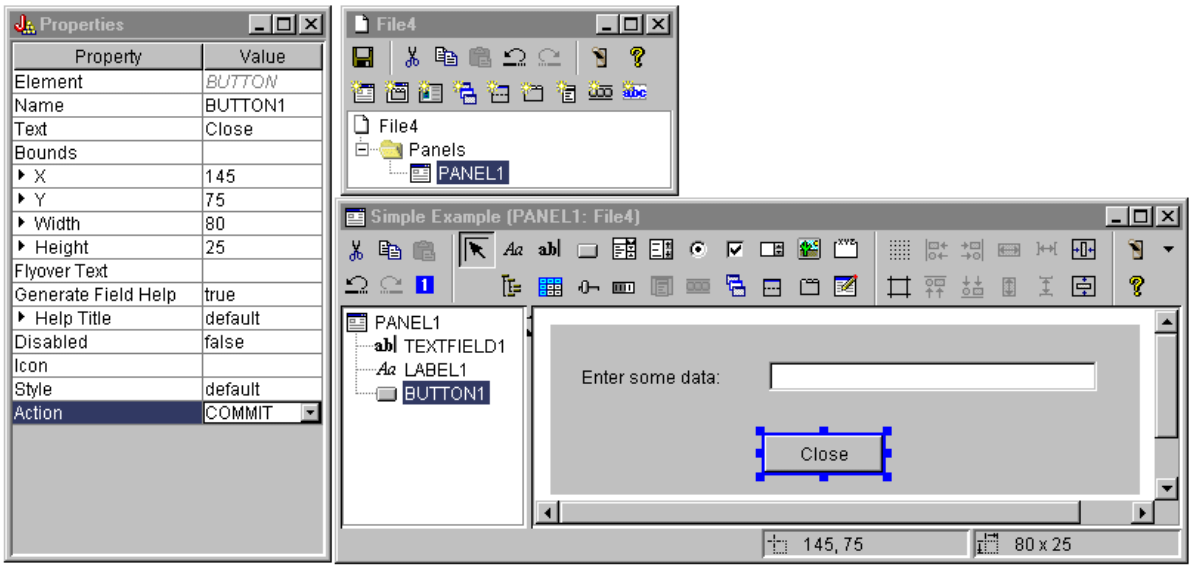
Modificare la proprietà di stile per assegnare al pulsante le caratteristiche predefinite.

Figura 7: finestre GUI Builder: impostazione della proprietà Stile per assegnare al pulsante le caratteristiche predefinite



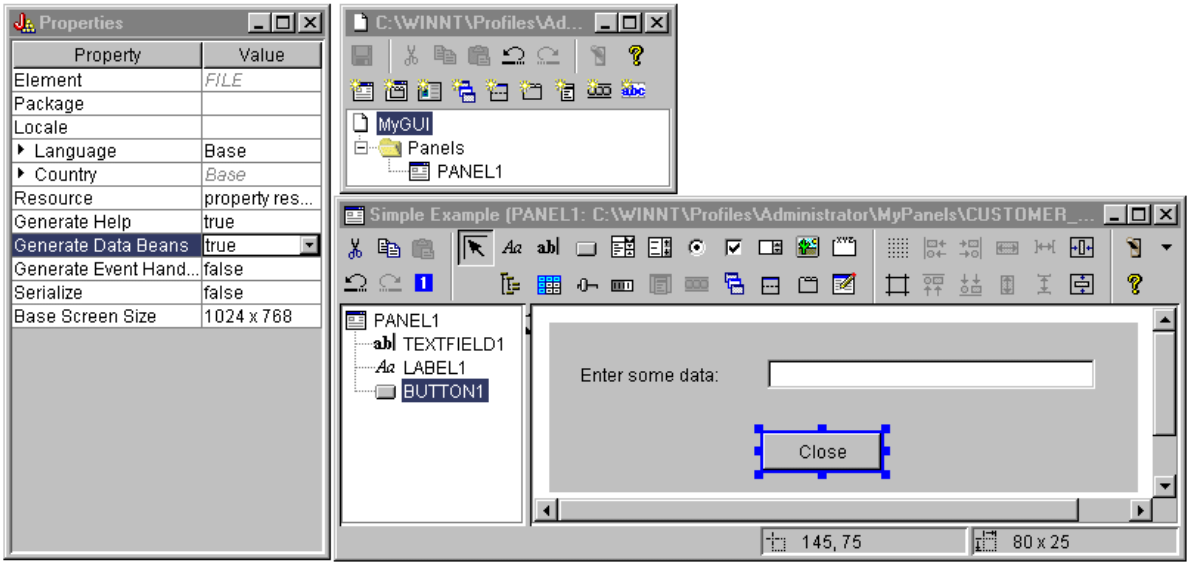
Impostare la proprietà ACTION su COMMIT, per fare in modo che il metodo `setUserData` sul bean venga chiamato quando viene selezionato il pulsante.

Figura 8: finestre GUI Builder: impostazione della proprietà ACTION su COMMIT



Prima di salvare il pannello, impostare le proprietà al livello del file PDML per creare sia la struttura dell'aiuto in linea che il java bean. Quindi, salvare il file facendo clic sull'icona nella finestra GUI Builder.Quando viene richiesto, specificare un nome file **MyGUI.pdml**.

Figura 9: finestre GUI Builder: impostazione delle proprietà per creare la struttura dell'aiuto in linea e il Javabean



File creati

Dopo aver salvato la definizione del pannello, è possibile esaminare i file creati dal GUI Builder. **File PDML** Questo è il contenuto di **MyGUI.pdml** per avere un'idea di come funziona il PDML. Poiché si utilizza PDML solo tramite gli strumenti forniti dal Graphical Toolbox, non è necessario conoscere il formato di questo file:

```
<!-- Creato dal GUI Builder -->
<PDML version="2.0" source="JAVA" basescreensize="1280x1024">

 <PANEL name="PANEL1">
  <TITLE>PANEL1<TITLE>
  <SIZE>351,162<</SIZE>
  <LABEL name="LABEL1"">
   <TITLE>PANEL1.LABEL1</TITLE>
   <LOCATION>18,36</LOCATION>
   <SIZE>94,18</SIZE>
   <HELPLINK>PANEL1.LABEL1</HELPLINK>
  </LABEL>
  <TEXTFIELD name="TEXTFIELD1">
   <TITLE>PANEL1.TEXTFIELD1</TITLE>
   <LOCATION>125,31</LOCATION>
   <SIZE>191,26</SIZE>
   <DATACLASS>SampleBean</DATACLASS>
   <ATTRIBUTE>UserData</ATTRIBUTE>
   <STRING minlength="0" maxlength="15"/>
   <HELPALIAS>LABEL1</HELPALIAS>
  </TEXTFIELD>
  <BUTTON name="BUTTON1">
   <TITLE>PANEL1.BUTTON1</TITLE>
   <LOCATION>125,100</LOCATION>
   <SIZE>100,26</SIZE>
   <STYLE>DEFAULT<</STYLE>
   <ACTION>COMMIT</ACTION>
   <HELPLINK>PANEL1.BUTTON1</HELPLINK>
  </BUTTON>
 </PANEL>

</PDML>
```

Bundle di risorse

Un bundle di risorse è associato ad ogni file PDML. In questo esempio, le risorse convertibili sono state salvate in un file PROPERTIES, che è denominato **MyGUI.properties**. Notare che il file PROPERTIES contiene anche dati di personalizzazione per il GUI Builder.

```
##Generated by GUI Builder
BUTTON_1=Close
TEXT_1=
@GenerateHelp=1
@Serialize=0
@GenerateBeans=1
LABEL_1=Enter some data:
PANEL_1.Margins=18,18,18,18,18,18
PANEL_1=Simple Example
```

JavaBean

Questo esempio crea anche una struttura di codice sorgente Java per l'oggetto JavaBean. Questo è il contenuto di **SampleBean.java**:

```
import com.ibm.as400.ui.framework.java.*;

public class SampleBean extends Object
    implements DataBean
{
    private String m_sUserData;

    public String getUserData()
    {
        return m_sUserData;
    }

    public void setUserData(String s)
    {
        m_sUserData = s;
    }

        public Capabilities getCapabilities()
    {
                return null;
    }

        public void verifyChanges()
    {
    }

        public void save()
    {
    }

        public void load()
    {
        m_sUserData ="";
    }
}
```

Notare che la struttura contiene già un'implementazione dei metodi gettor e settor per la proprietà `UserData`.Gli altri metodi sono definiti dall'interfaccia `DataBean` e, quindi, sono necessari.

Il GUI Builder ha già richiamato il programma di compilazione Java per la struttura e creato il file di classe corrispondente. Per lo scopo di questo semplice esempio, non è necessario modificare l'implementazione del bean. In un'applicazione reale Java si modificano i metodi `load` e `save` per trasferire i dati da un sorgente dati esterno. L'implementazione predefinita degli altri due metodi è spesso sufficiente.Per ulteriori informazioni, consultare la documentazione sull'interfaccia **DataBean** nel javadoc per la framework tempo di esecuzione PDML.

File di aiuto

Il GUI Builder crea anche una framework HTML denominata Documento di aiuto. I programmi di scrittura di aiuto possono gestire facilmente informazioni di aiuto modificando questo file. Per ulteriori informazioni, consultare i seguenti argomenti:

- Creazione del documento aiuto
- Modifica dei documenti di aiuto creati dal GUI Builder

Creazione dell'applicazione

Una volta che la definizione del pannello ed i file generati sono stati salvati, si è pronti per creare l'applicazione. Tutto ciò di cui si ha bisogno è un file sorgente Java che conterrà il punto di immissione principale per l'applicazione. Per questo esempio, il file è denominato **SampleApplication.java**. Esso contiene il seguente codice:

```
import com.ibm.as400.ui.framework.java.*;
import java.awt.Frame;

public class SampleApplication{
        public static void main (String[] args)
    {
                // Avviare l'oggetto bean che fornisce i dati al pannello
                SampleBean bean = new SampleBean();

                // Inizializzare l'oggetto
                bean.load();

                // Impostare l'inoltro del bean al gestore pannelli
                DataBean[] beans = { bean };

        // Creare il gestore pannelli.
Parametri:
                // 1. File PDML come nome risorsa
                // 2. Nome del pannello da visualizzare
                // 3. Elenco degli oggetti di dati che forniscono i dati al pannello
        // 4. Una frame AWT per rendere modale il pannello

        PanelManager pm = null;
        try { pm = new PanelManager("MyGUI", "PANEL_1", beans, new Frame()); }
                catch (DisplayManagerException e)
        {
                        // Si è verificato un errore, quindi visualizzare un messaggio e uscire
                        e.displayUserMessage(null);
                        System.exit(1);
        }

        // Visualizzare il pannello - il controllo viene interrotto qui
        pm.setVisible(true);

        // Ripetere i dati utente salvati
        System.out.println("SAVED USER DATA: '" + bean.getUserData() + "'");

        // Uscire dall'applicazione
                System.exit(0);
    }
}
```

E' compito del programma di chiamata inizializzare l'oggetto o gli oggetti bean chiamando **load**. Se i dati per un pannello vengono forniti da più oggetti bean, è necessario inizializzare ognuno di questi oggetti prima di passarli all'ambiente Graphical Toolbox.

La classe **com.ibm.as400.ui.framework.java.PanelManager** fornisce l'API per visualizzare finestre e finestre di dialogo autonome. Il nome del file PDML come viene fornito nel programma di creazione,

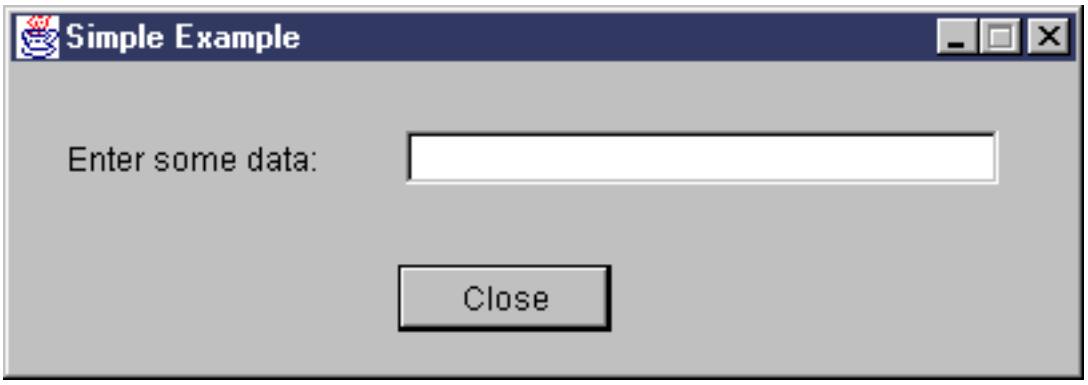
viene considerato come nome della risorsa dal Graphical Toolbox, l'indirizzario, il file ZIP o il file JAR contenenti il PDML devono essere identificati nel classpath.

Poiché un oggetto **Frame** è fornito dal programma di creazione, la finestra si comporterà come una finestra di dialogo modale. In un'applicazione Java reale, questo oggetto può essere ottenuto da una finestra principale appropriata per la finestra di dialogo. Dal momento che la finestra è modale, il controllo non restituisce l'applicazione fino a quando l'utente non chiude la finestra. A quel punto, l'applicazione semplicemente replica i dati utente modificati ed esce.

Esecuzione di un'applicazione

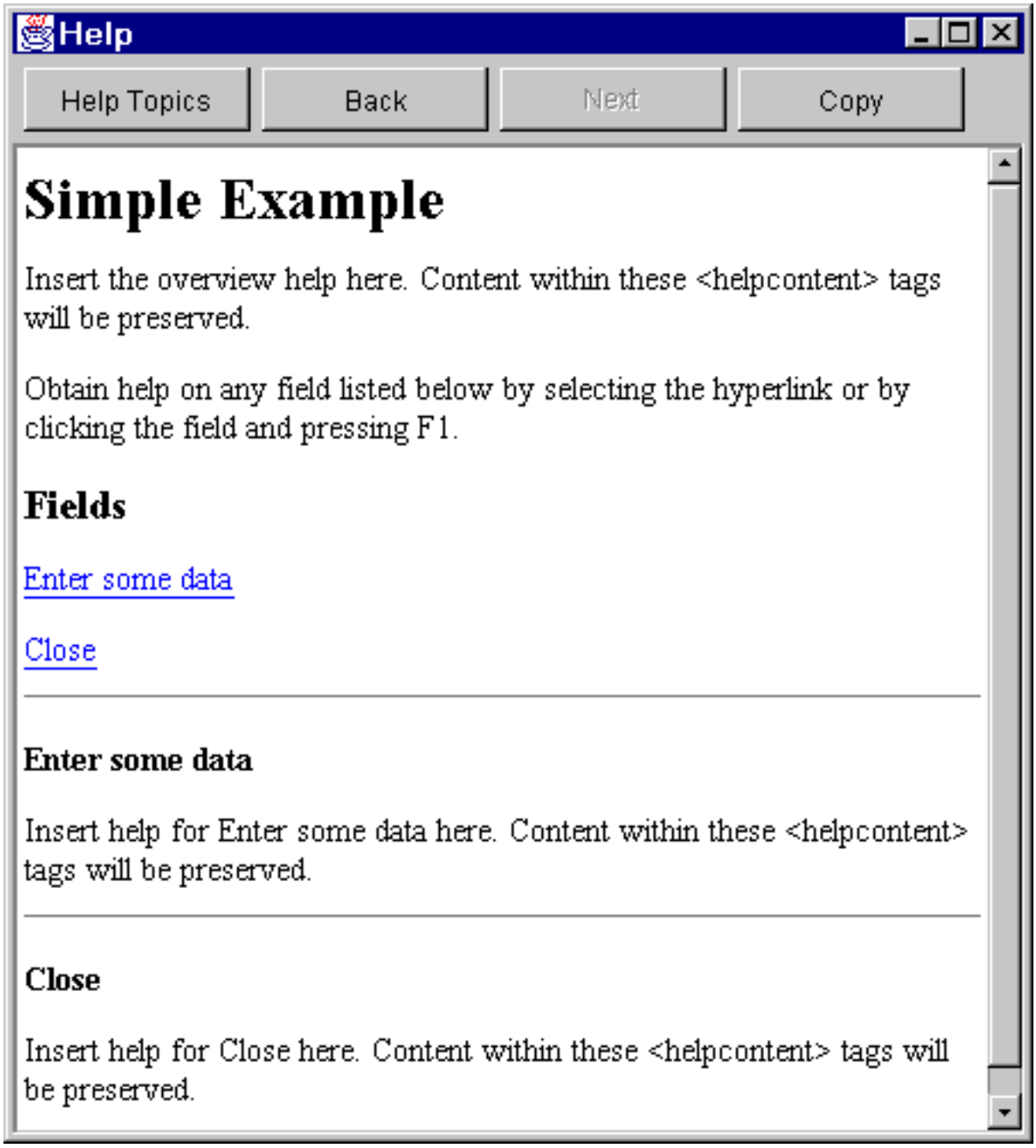
Questo è un esempio di come appare la finestra quando l'applicazione viene compilata ed eseguita:

Figura 10: finestra dell'applicazione Esempio semplice



Se l'utente preme F1 mentre è evidenziato il campo testo, il Graphical Toolbox visualizzerà un browser di aiuto contenente la struttura dell'aiuto in linea creata dal GUI Builder.

Figura 11: struttura dell'aiuto in linea dell'Esempio semplice



E' possibile modificare l'HTML ed aggiungere il contenuto effettivo dell'aiuto per gli argomenti di aiuto mostrati.

Se i dati nel campo testo non sono validi (ad esempio, se l'utente ha fatto clic sul pulsante **Chiudi** senza fornire un valore), il Graphical Toolbox visualizzerà un messaggio di errore ed evidenzierà nuovamente il campo in modo che i dati possano essere immessi.

Figura 12: messaggio errore dati

Per informazioni su come eseguire questo esempio come un'applet, consultare Utilizzo di Graphical Toolbox in un Browser.

Caselle combinate modificabili

Quando il generatore bean crea un programma di ricezione ed impostazione per una casella combinata modificabile, per impostazione predefinita esso restituisce una stringa sul programma di impostazione e richiede un parametro di stringa sul programma di ricezione. Può essere utile modificare il programma di impostazione per ricevere una classe Oggetto e il programma di ricezione per restituire un tipo Oggetto. Questo consente di determinare la selezione dell'utente utilizzando ChoiceDescriptors.

Se un tipo di Oggetto è rilevato per il programma di ricezione e di impostazione, il sistema prevede un ChoiceDescriptor o un tipo Oggetto invece di una stringa formattata.

Il seguente esempio, presume che Editable sia una Casella combinata modificabile con un valore doppio, che utilizzi un valore di sistema o che non sia impostata.

```
public Object getEditable()
{
        if (m_setting == SYSTEMVALUE)
    {
            return new ChoiceDescriptor("choice1","System Value");
    }
        else if (m_setting == NOTSET)
    {
            return new ChoiceDescriptor("choice2","Value not set");
    }
                        else
    {
            return m_doubleValue;
    }
}
```

Analogamente, quando un tipo di Oggetto è rilevato per il programma di ricezione ed impostazione, il sistema restituisce un Oggetto che è un ChoiceDescriptor contenente le scelte selezionate o un tipo di Oggetto.

```
     public void setEditable(Object item)
     {
             if (ChoiceDescriptor.class.isAssignableForm(obj.getClass()))
         {
            if (((ChoiceDescriptor)obj).getName().equalsIgnoreCase("choice1"))
                        m_setting = SYSTEMVALUE;
                                else
                        m_setting = NOTSET;
         }
             else if (Double.class.isAssignableFrom(obj.getClass()))
         {
                m_setting = VALUE;
                m_doubleValue = (Double)obj;
```

```
        }
                              else
            { /* error processing */ }
    }
```

Esempio: utilizzo di RecordListFormPane

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
///////////////////////////////////////////////////////////////////////////
//
// Esempio di RecordListFormPane.  Questo programma presenta un modulo con
// il contenuto di un file nel server.
//
// Sintassi del comando:
//    RecordListFormPaneExample system fileName
//
// Questo sorgente è un esempio di "RecordListFormPane" di IBM Toolbox per Java.
//
///////////////////////////////////////////////////////////////////////////

import com.ibm.as400.access.*;
import com.ibm.as400.vaccess.*;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class RecordListFormPaneExample
{


        public static void main (String[] args)
    {
        // Se non è stato specificato un sistema e un fileName, visualizzare
        // il testo di aiuto ed uscire.
                if (args.length != 2)
        {
            System.out.println("Usage:  RecordListFormPaneExample system fileName");
                        return;
        }

                                    try
        {
            // Creare un oggetto AS400.
Il nome di sistema è stato passato
            // come primo argomento della riga comandi.
                        AS400 system = new AS400 (args[0]);

            // Creare una frame.
            JFrame f = new JFrame ("RecordListFormPane example");

            // Creare un adattatore finestra di dialogo errore.  Questo visualizzerà
            // qualsiasi errore all'utente.
                        ErrorDialogAdapter errorHandler = new ErrorDialogAdapter (f);

            // Creare un pannello modulo elenco record per presentare il contenuto
            // del database.  Si noti che viene creato il pannello modulo, aggiunto
            // il listener degli errori, quindi impostato il sistema ed il nome file.
            // La creazione del pannello modulo e l'impostazione dei relativi parametri
            // possono essere eseguite in una sola fase nel modo seguente:
            //    RecordListFormPane formPane = new RecordListFormPane (system, args[1]);
            // Il potenziale problema è costituito dal fatto che non vi è ancora un listener degli errori
            // quindi se il nome file non è corretto, non vi è alcuna ubicazione
            // nella quale visualizzare l'errore.
                        RecordListFormPane formPane = new RecordListFormPane();
                    formPane.addErrorListener (errorHandler);
                        formPane.setSystem(system);
```

```
                formPane.setFileName(args[1]);

        // Richiamare le informazioni dal sistema.
                formPane.load ();

        // Quando si chiude la frame, uscire.
                f.addWindowListener (new WindowAdapter () {
                        public void windowClosing (WindowEvent event)
            {
                  System.exit(0);
            }
        });

        // Effettuare il layout della frame con il pannello modulo.
                f.getContentPane ().setLayout (new BorderLayout ());
        f.getContentPane ().add ("Center", formPane);
                f.pack ();
                f.show ();
    }
                            catch (Exception e)
    {
                System.out.println ("Error: " + e.getMessage ());
             System.exit(0);
    }
}

}
```

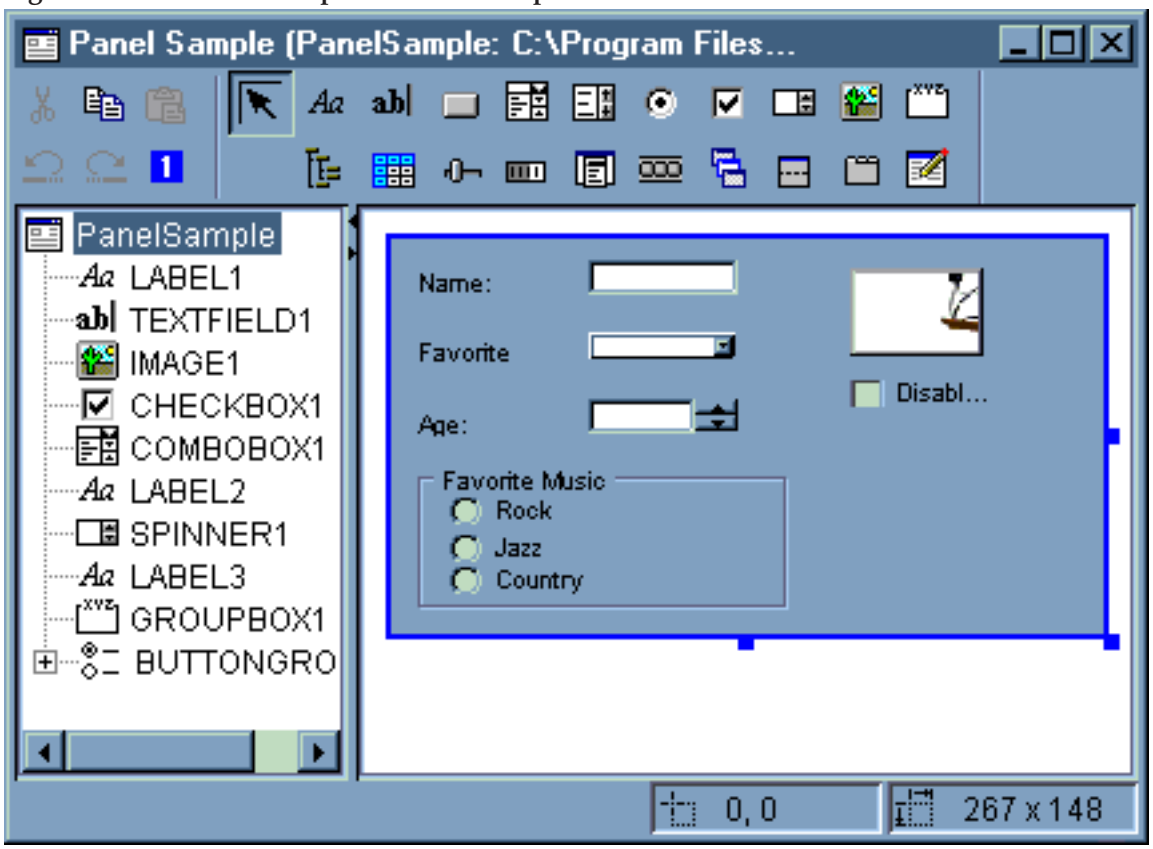
Creazione di un pannello con il GUI Builder

La creazione di un pannello con il GUI Builder è semplice. Dalla barra dei menu nella finestra principale del GUI Builder, selezionare **File** → **Nuovo File**.

Dalla barra dei menu nella finestra **File** del GUI Builder, fare clic sull'icona Inserisci nuovo pannello  per visualizzare un programma di creazione del pannello in cui è possibile inserire i componenti del pannello.I pulsanti della barra degli strumenti sulla finestra **Pannello** rappresentano diversi componenti che possono essere aggiunti al pannello. Selezionare il componente desiderato, quindi fare clic sul punto in cui si desidera posizionarlo.

La figura che segue mostra un pannello creato con molte delle opzioni disponibili.

Figura 1: creazione di un pannello di esempio con GUI Builder



Il pannello di esempio nella Figura 1 utilizza il seguente codice DataBean per raccogliere i diversi componenti:

```
import com.ibm.as400.ui.framework.java.*;

public class PanelSampleDataBean extends Object
    implements DataBean
{
        private String m_sName;
        private Object m_oFavoriteFood;
        private ChoiceDescriptor[] m_cdFavoriteFood;
        private Object m_oAge;
        private String m_sFavoriteMusic;

        public String getName()
    {
                return m_sName;
    }

        public void setName(String s)
    {
                m_sName = s;
    }

        public Object getFavoriteFood()
    {
                return m_oFavoriteFood;
    }

        public void setFavoriteFood(Object o)
    {
                m_oFavoriteFood = o;
    }
```

```
        public ChoiceDescriptor[] getFavoriteFoodChoices()
    {
            return m_cdFavoriteFood;
    }

        public Object getAge()
    {
            return m_oAge;
    }

        public void setAge(Object o)
    {
            m_oAge = o;
    }

        public String getFavoriteMusic()
    {
            return m_sFavoriteMusic;
    }

        public void setFavoriteMusic(String s)
    {
            m_sFavoriteMusic = s;
    }

        public Capabilities getCapabilities()
    {
        return null;
    }

        public void verifyChanges()
    {
    }

        public void save()
    {
            System.out.println("Name = " + m_sName);
            System.out.println("Favorite Food = " + m_oFavoriteFood);
            System.out.println("Age = " + m_oAge);
            String sMusic = "";
            if (m_sFavoriteMusic != null)
        {
                if (m_sFavoriteMusic.equals("RADIOBUTTON1"))
                        sMusic = "Rock";
                else if (m_sFavoriteMusic.equals("RADIOBUTTON2"))
                        sMusic = "Jazz";
                else if (m_sFavoriteMusic.equals("RADIOBUTTON3"))
                        sMusic = "Country";
        }
            System.out.println("Favorite Music = " + sMusic);
    }

        public void load()
    {
            m_sName = "Sample Name";
            m_oFavoriteFood = null;
            m_cdFavoriteFood = new ChoiceDescriptor[0];
            m_oAge = new Integer(50);
            m_sFavoriteMusic = "RADIOBUTTON1";
    }
}
```

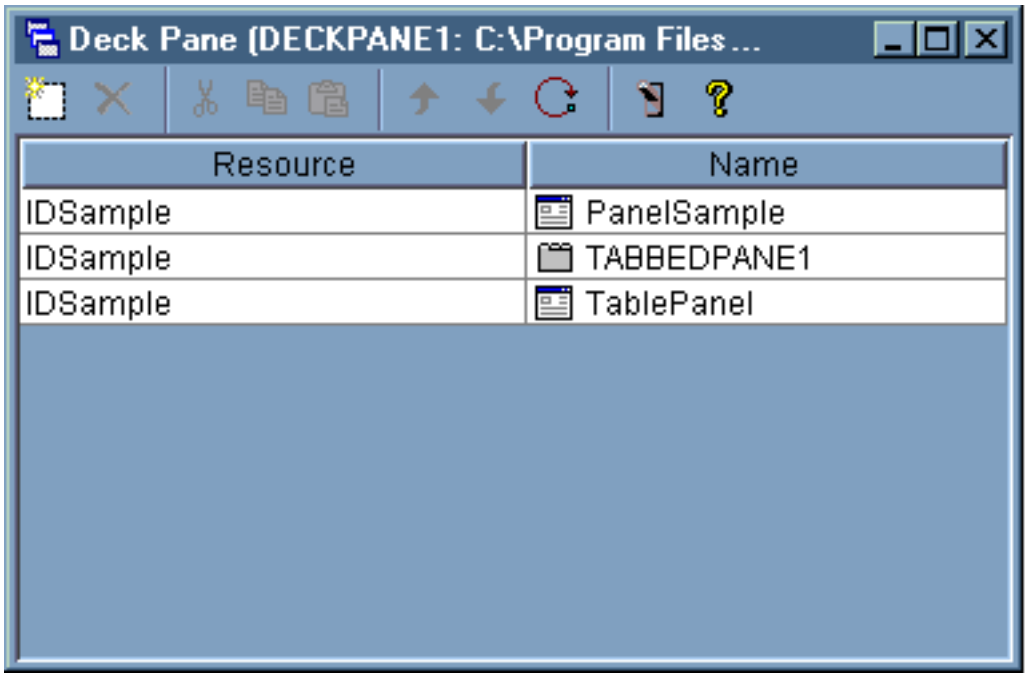
Il pannello è il componente più semplice disponibile nel GUI Builder, ma da un semplice pannello è possibile creare notevoli applicazioni UI.

Creazione di un pannello sovrapposto con il GUI Builder

Il GUI Builder crea un semplice pannello suddiviso. Dalla barra dei menu nella finestra GUI Builder, selezionare **File** --> **Nuovo File**.

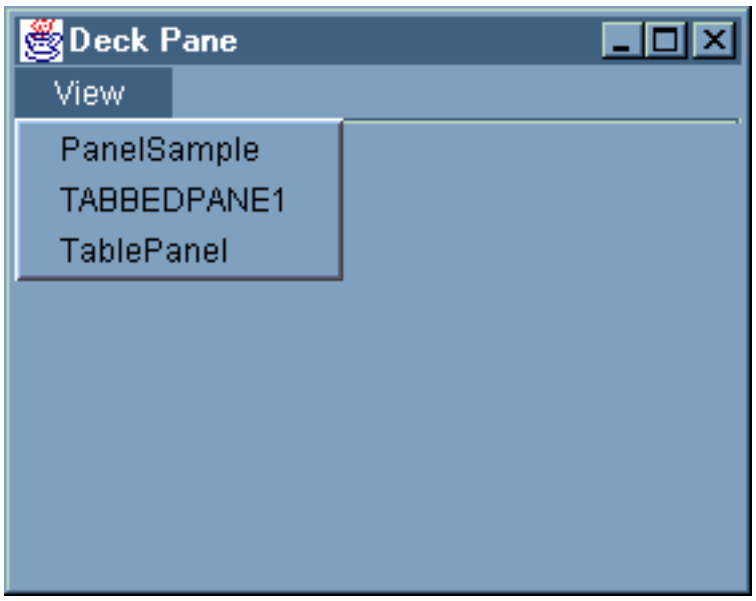
Dalla barra dei menu nella finestra **File** del GUI Builder, fare clic sul pulsante strumento **Inserisci pannello suddiviso**  per visualizzare un programma di creazione del pannello in cui è possibile inserire i componenti per il pannello suddiviso. Nell'esempio seguente, vengono aggiunti tre componenti.

Figura 1: creazione di un pannello suddiviso con il GUI Builder



Dopo aver creato il pannello suddiviso, fare clic sul pulsante strumento **Anteprima**  per visualizzarlo in anteprima. Viene visualizzato un pannello suddiviso fino a che non si seleziona il menu **Visualizza**.

Figura 2: anteprima del pannello suddiviso con il GUI Builder



Dal menu **Visualizza** del pannello suddiviso, selezionare il componente che si desidera visualizzare. Per questo esempio, si può scegliere di visualizzare il PanelSample, TABBEDPANE1 o il TablePanel. Le seguenti figure illustrano come vengono visualizzati questi componenti.

Figura 3: visualizzazione di PanelSample con il GUI Builder

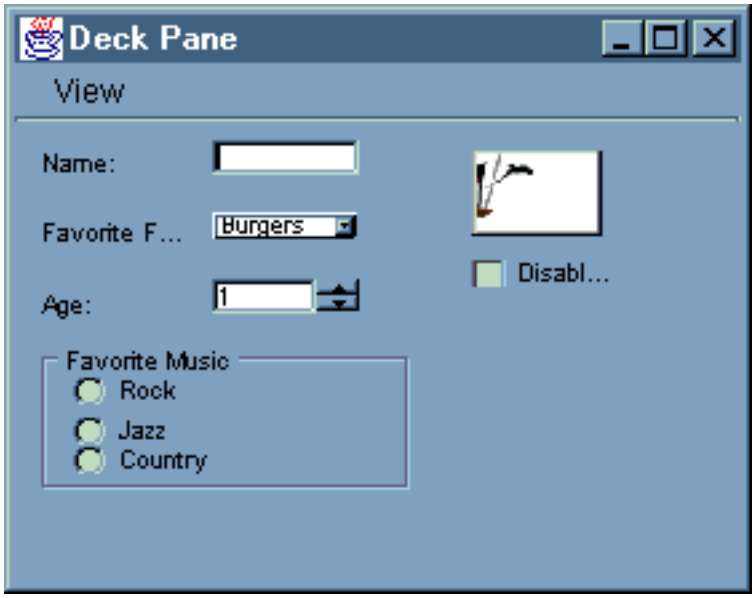


Figura 4: visualizzazione di TABBEDPANE1 con il GUI Builder

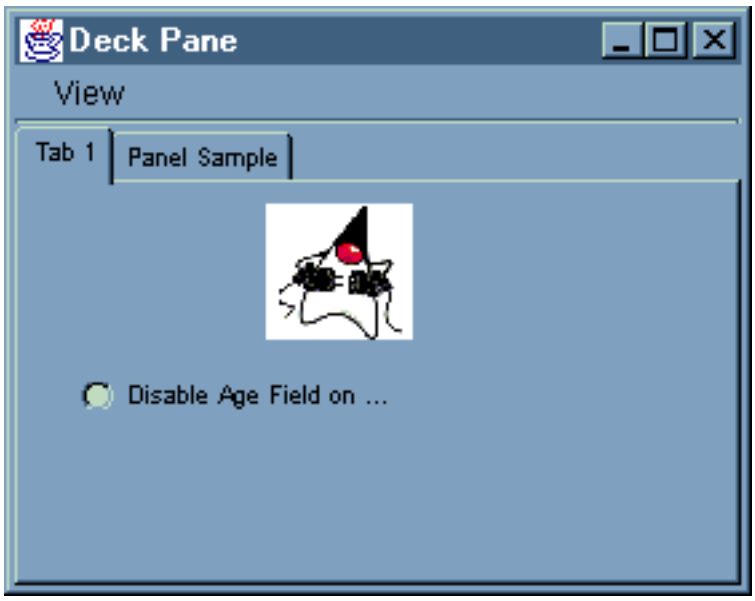
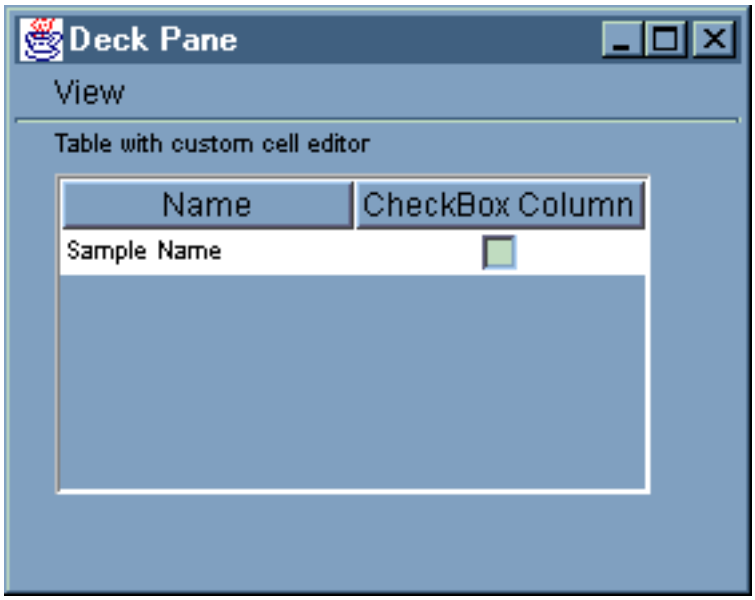


Figura 5: visualizzazione di TablePanel con il GUI Builder

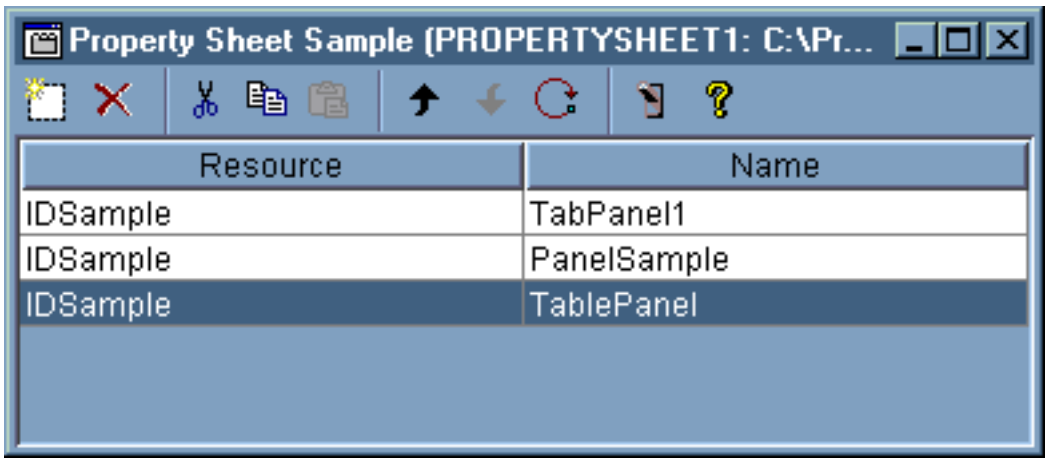


Creazione di un foglio delle proprietà con il GUI Builder

Il GUI Builder semplifica la creazione di un foglio delle proprietà. Dalla barra dei menu nella finestra principale di GUI Builder, selezionare **File** --> **Nuovo file**.

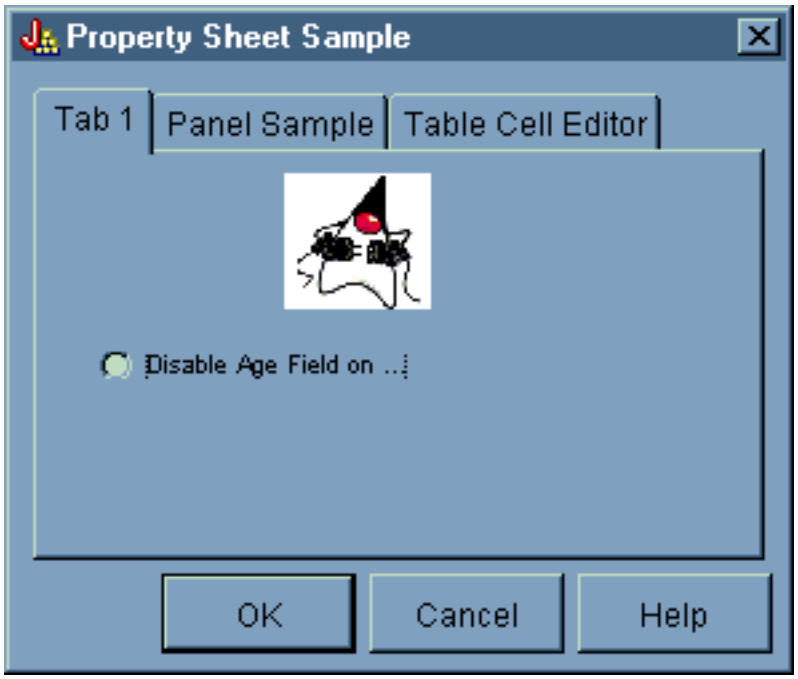
Dalla barra dei menu nella finestra **File** del GUI Builder, fare clic sull'icona Inserisci foglio proprietà per visualizzare un programma di creazione del pannello in cui si è possibile inserire i componenti del foglio delle proprietà.

Figura 1: creazione di un foglio delle proprietà con il GUI Builder



Dopo aver creato il foglio delle proprietà, utilizzare l'icona per visualizzarlo in anteprima.Per tale esempio, è possibile effettuare una scelta fra tre separatori.

Figura 2: visualizzazione in anteprima di un foglio delle proprietà con il GUI Builder

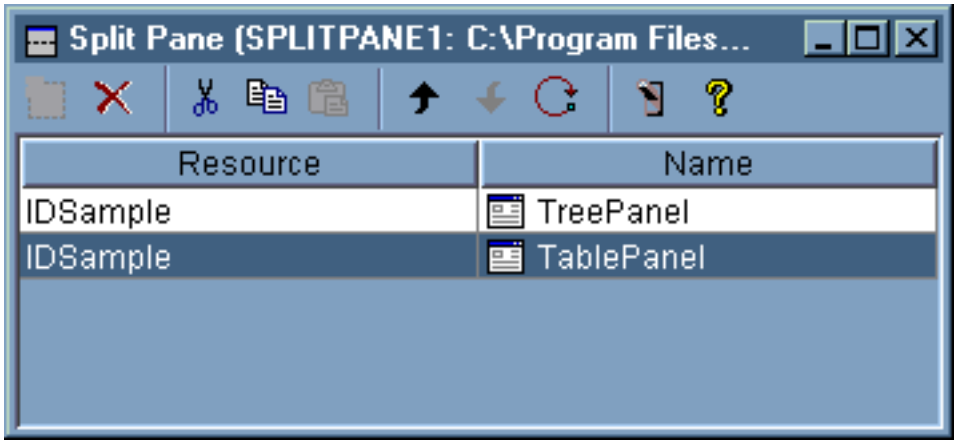


Creazione di un pannello suddiviso con il GUI Builder

Il GUI Builder fa si che la creazione di un pannello suddiviso sia semplice. Dalla barra dei menu nella finestra principale di GUI Builder, selezionare **File** --> **Nuovo file**.

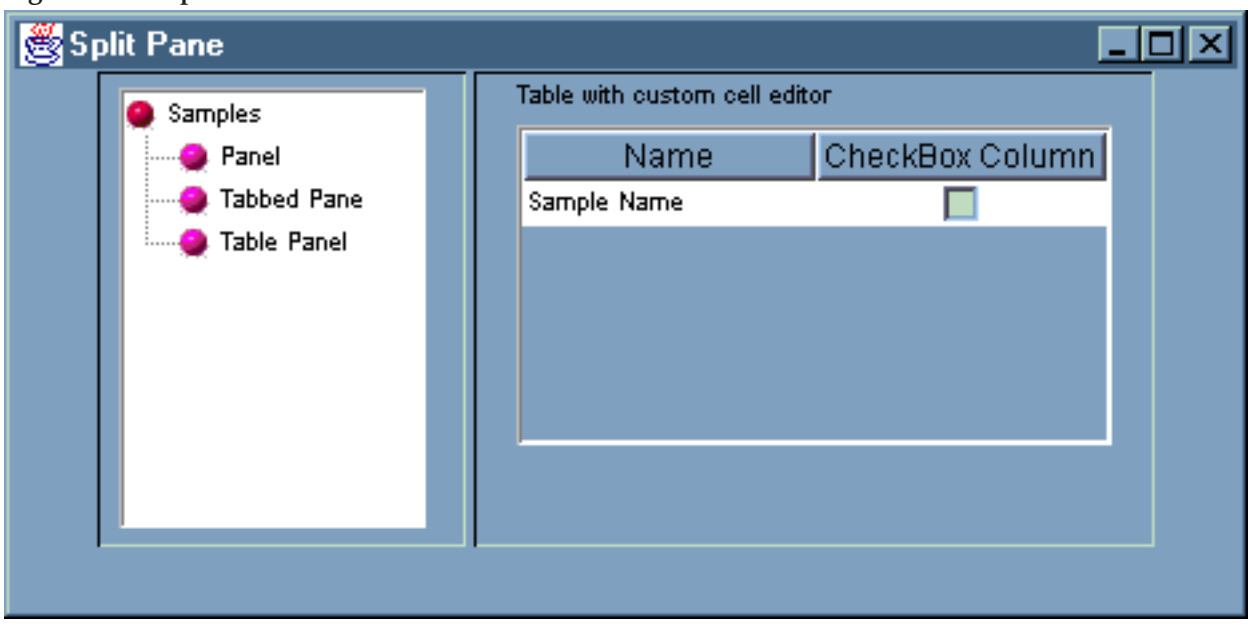
Dalla barra dei menu della finestra **File** del GUI Builder, fare clic sul pulsante strumento Inserisci pannello suddiviso per visualizzare un programma di creazione del pannello in cui è possibile inserire i componenti desiderati. Nell'esempio che segue, vengono aggiunti due componenti.

Figura 1: creazione di un pannello suddiviso con il GUI Builder



Una volta creato il pannello suddiviso, fare clic sull'icona del pulsante dello strumento **Anteprima** per visualizzarlo, come mostrato in Figura 2.

Figura 2: Anteprima del Pannello suddiviso con il GUI Builder

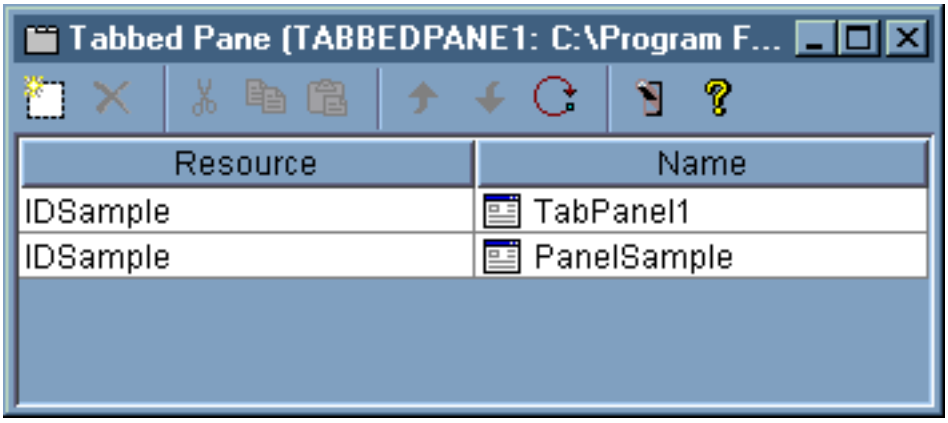


Creazione di un pannello con separatori con il GUI Builder

Il GUI Builder fa sì che la creazione di un pannello con separatori sia semplice. Dalla barra dei menu nella finestra principale di GUI Builder, selezionare **File** --> **Nuovo file**.

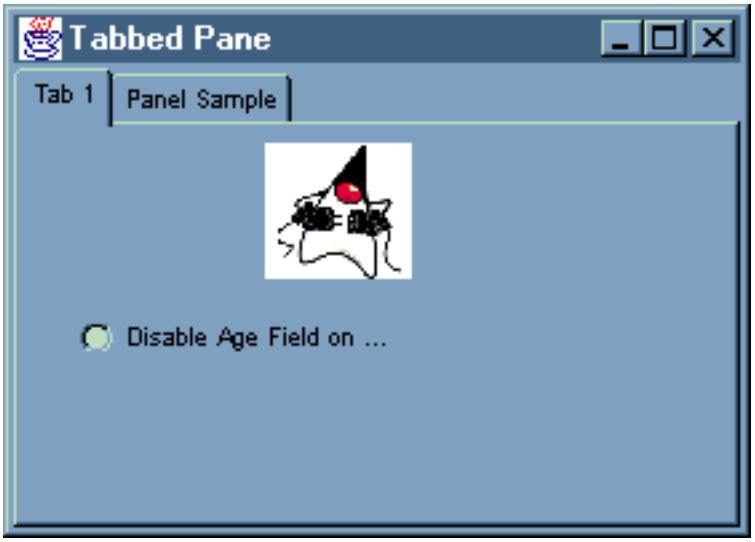
Dalla barra dei menu nella finestra **File** del Gui Builder, fare clic sull'icona Inserisci pannello con separatori per visualizzare un programma di creazione del pannello in cui è possibile inserire i componenti per il pannello con separatori. Nell'esempio che segue, vengono aggiunti due componenti.

Figura 1: creazione di un pannello con separatori nel GUI Builder



Dopo aver creato il pannello con separatori, fare clic sul pulsante dello strumento **Anteprima** per avere una visualizzazione in anteprima di esso.

Figura 2: anteprima del pannello con separatori con il GUI Builder



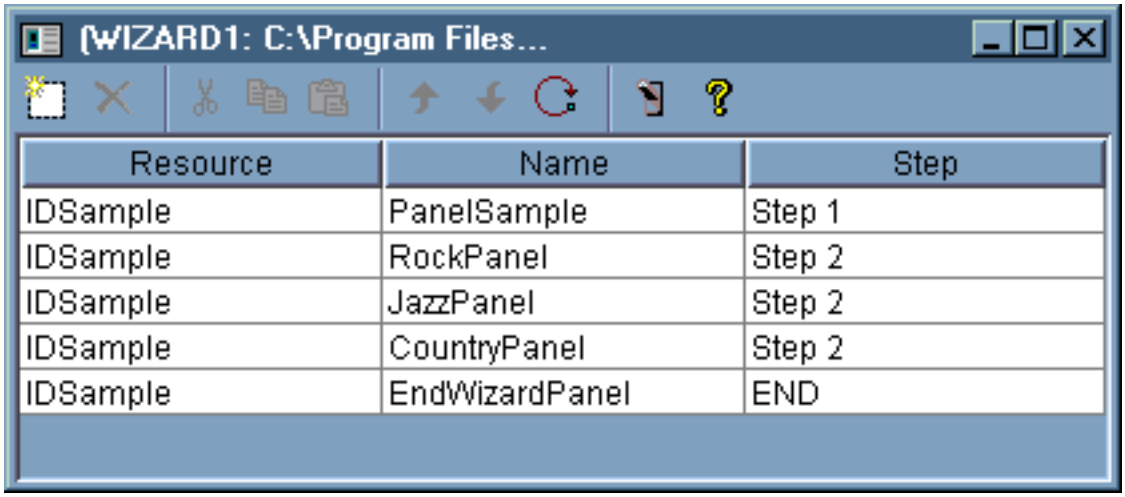
Creazione di un wizard con il GUI Builder

Il GUI Builder semplifica la creazione di un'interfaccia wizard. Dalla barra di menu sulla finestra GUI Builder, selezionare **File** --> **Nuovo File**.

Dalla barra dei menu sulla finestra **File** del GUI Builder, fare clic sul pulsante della barra degli Inserisci wizard

per visualizzare un programma di creazione del pannello da cui è possibile aggiungere pannelli al wizard.

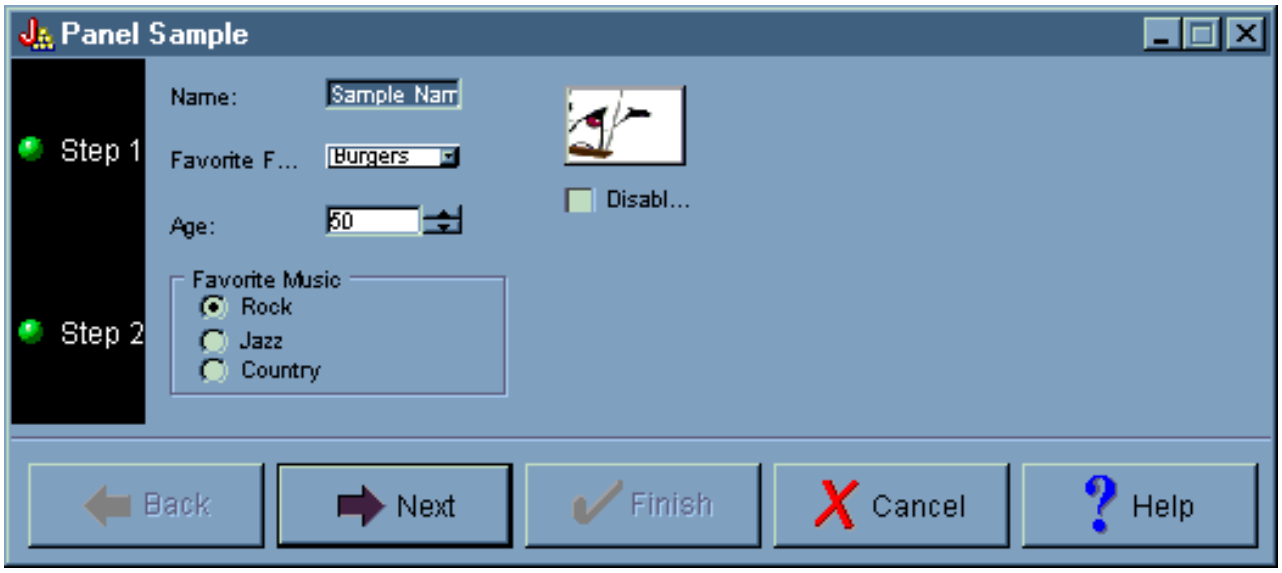
Figura 1: creazione di un Wizard con il GUI Builder



Dopo aver creato il wizard, utilizzare il pulsante dello strumento **Anteprima**

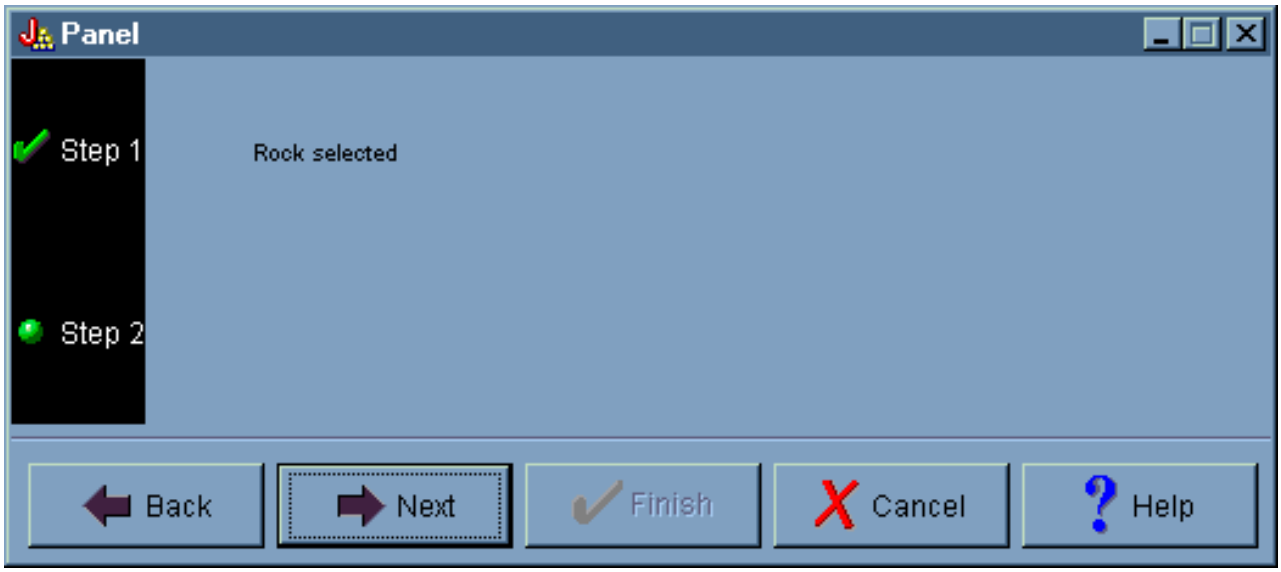
per avere una visualizzazione in anteprima di esso. La figura 2 mostra il primo pannello visualizzato per questo esempio.

Figure 2: anteprima del primo pannello del wizard con il GUI Builder



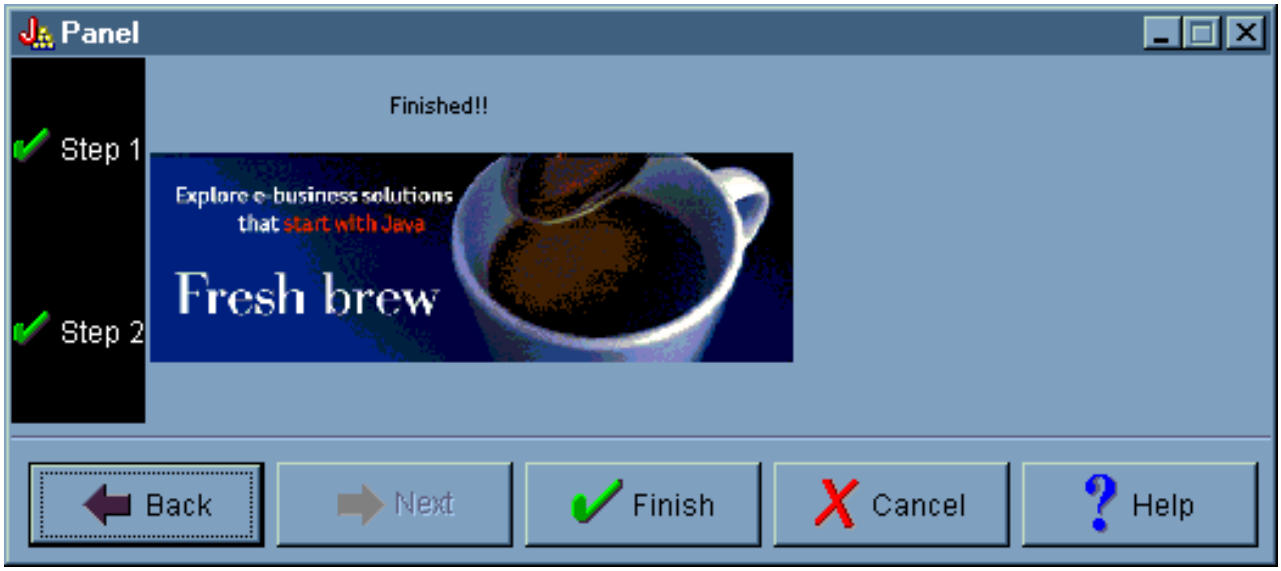
La figura 2 mostra il secondo pannello visualizzato quando l'utente seleziona **Rock** e fa clic su **Avanti**.

Figure 3: Anteprima del secondo pannello del wizard con il GUI Builder



Facendo clic su **Avanti** sul secondo pannello del wizard si visualizza il pannello finale del wizard, come mostrato nella Figura 4.

Figura 4: Anteprima del pannello finale del wizard con il GUI Builder

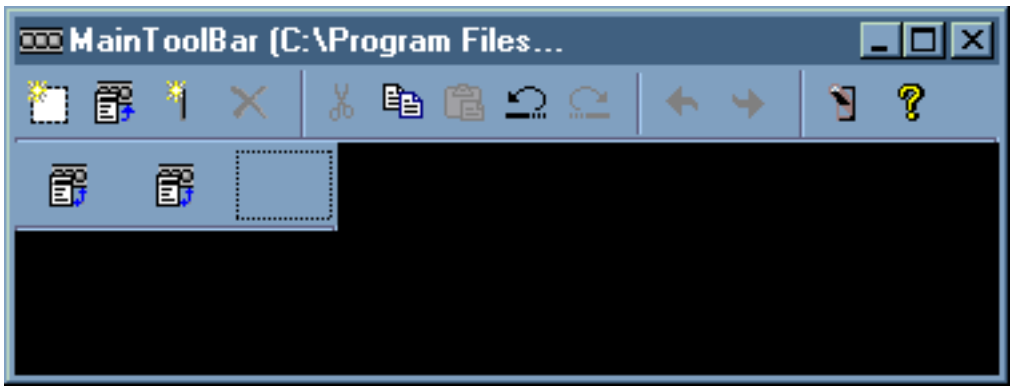


Creazione di una barra degli strumenti con il GUI Builder

Il GUI Builder rende semplice la creazione di una barra degli strumenti. Dalla barra di menu sulla finestra GUI Builder, selezionare **File** --> **Nuovo File**.

Dalla barra dei menu sulla finestra **File** del GUI Builder, fare clic sul pulsante strumento **Inserisce barra degli strumenti** per visualizzare il programma di creazione del pannello in cui è possibile inserire i componenti per la barra degli strumenti.

Figura 1: creazione di una barra degli strumenti con il GUI Builder



Dopo aver creato la barra degli strumenti, fare clic sul pulsante dello strumento **Anteprima** per avere una visualizzazione in anteprima di essa. Per questo esempio, è possibile visualizzare il foglio delle proprietà o il wizard.

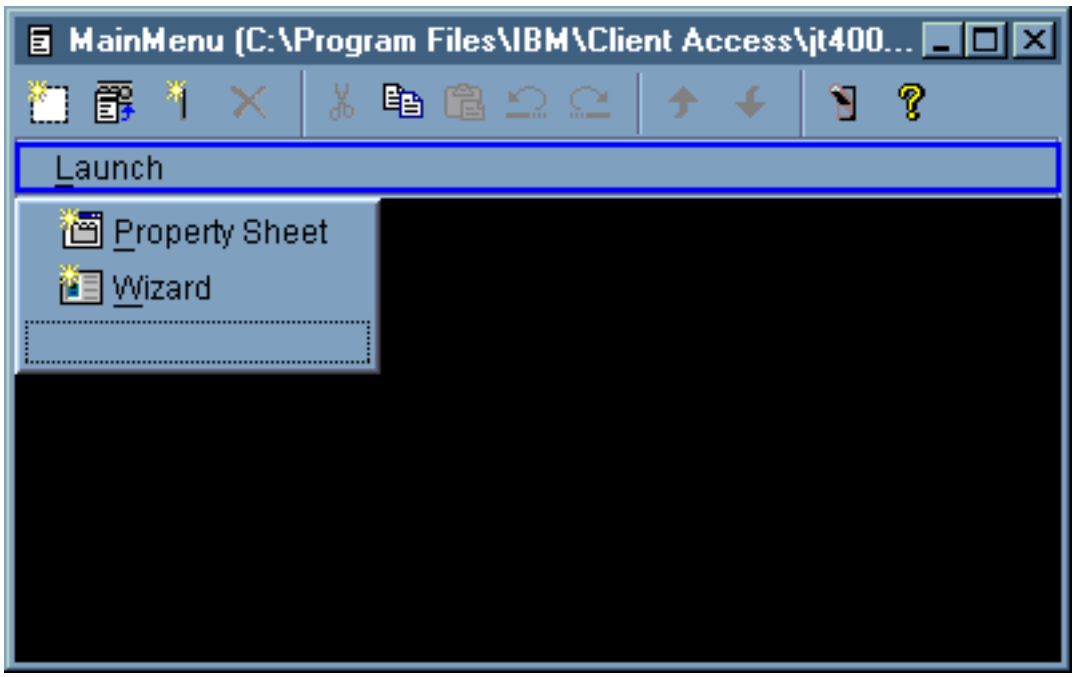
Figure 2: anteprima della barra degli strumenti con il GUI Builder

Creazione della barra dei menu con il GUI Builder

Il GUI Builder semplifica la creazione di una barra dei menu. Dalla barra dei menu nella finestra GUI Builder, selezionare **File** --> **Nuovo File**.

Dalla barra degli strumenti della finestra **File** di GUI Builder, fare clic sul pulsante strumento **Inserisci Menu** per creare un programma di creazione del pannello dove è possibile inserire i componenti per il menu.

Figura 1: GUI Builder: creazione di un Menu



Dopo aver creato il menu, utilizzare il pulsante strumento **Anteprima** per visualizzarlo in anteprima. Per questo esempio, dal menu **Avvio** appena creato è possibile selezionare **Foglio delle proprietà** o **Wizard**. Le seguenti figure illustrano cosa si visualizza quando vengono selezionati questi elementi del menu.

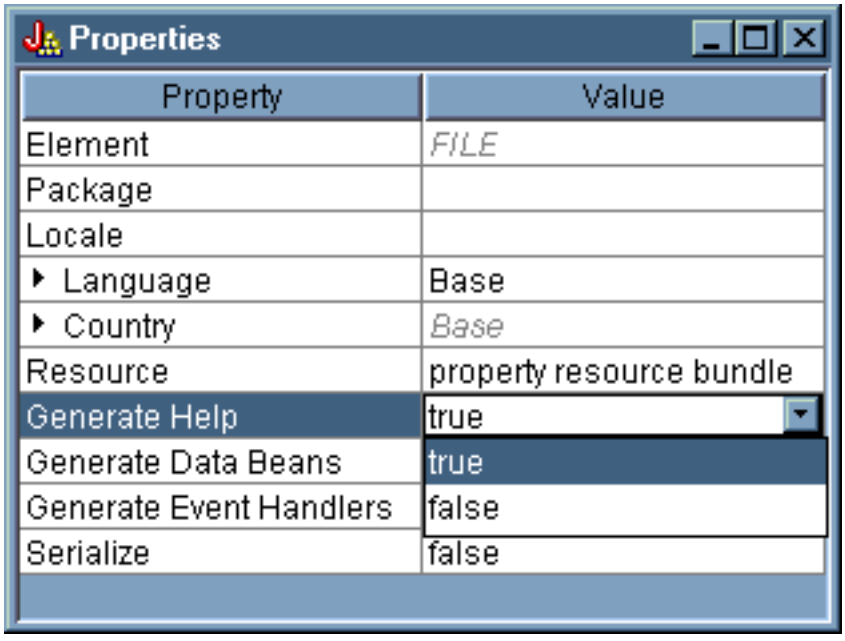
Figura 2: GUI Builder: visualizzazione del Foglio delle proprietà sul menu Avvio

Figura 3: GUI Builder: visualizzazione del Wizard sul menu Avvio

Esempio: creazione del documento di aiuto

Creare file di aiuto con GUI Builder è semplice. Sul pannello delle proprietà per il file utilizzato, impostare "Crea aiuto" su true.

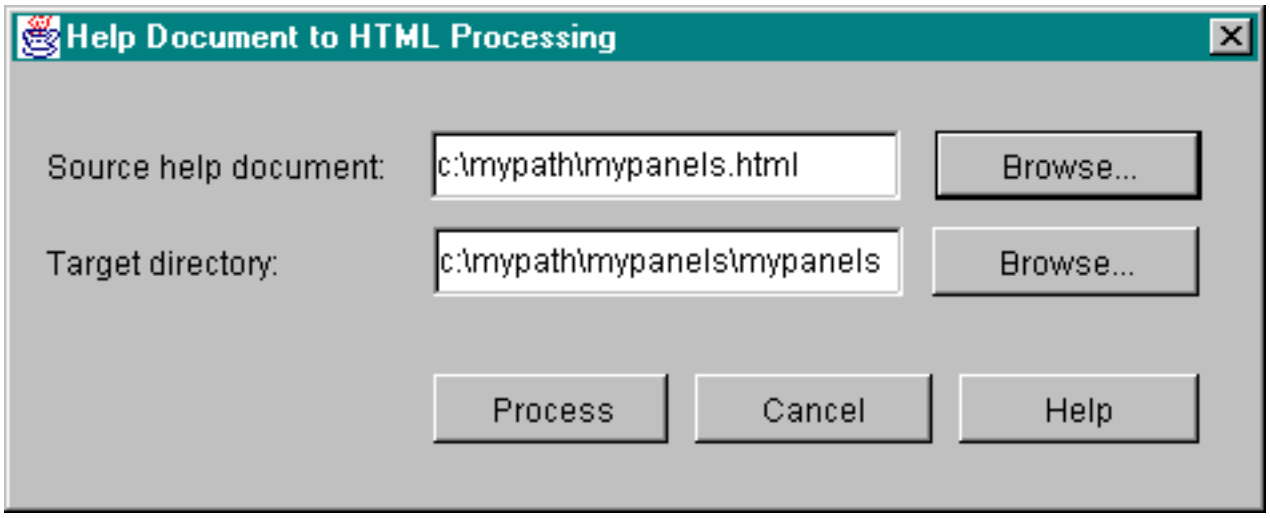
Figura 1: impostazione della proprietà Crea aiuto sul pannello delle proprietà del GUI Builder



GUI Builder crea una framework HTML denominata Documento di aiuto, che è possibile modificare.

Per fare in modo che venga utilizzato al momento dell'esecuzione, è necessario suddividere in file individuali HTML gli argomenti contenuti nel file PDML. Ogni volta che si esegue il **Documento di aiuto per l'elaborazione HTML**, gli argomenti vengono divisi in file singoli e immessi in un sottoindirizzario con lo stesso nome del Documento di aiuto e del file PDML. L'ambiente del tempo di esecuzione prevede che i file HTML singoli si trovino in un sottoindirizzario con lo stesso nome del Documento di aiuto e del file PDML. La finestra di dialogo **Documento di aiuto per l'elaborazione HTML** raccoglie le informazioni necessarie e richiama il programma HelpDocSplitter per eseguire l'elaborazione:

Figura 2: finestra di dialogo Documento di aiuto per l'elaborazione HTML



Il Documento di aiuto nell'elaborazione HTML viene avviato da una richiesta comandi, immettendo:

```
jre com.ibm.as400.ui.tools.hdoc2htmViewer
```

L'esecuzione di questo comando richiede che il percorso di classe sia impostato correttamente.

Per utilizzare il Documento di aiuto per l'elaborazione HTML, è necessario selezionare in primo luogo il Documento di aiuto con lo stesso nome del file PDML. Quindi, si specifica un sottoindirizzario con lo stesso nome del Documento di aiuto e del file PDML per l'emissione. Selezionare ″Elabora″ per completare l'elaborazione.

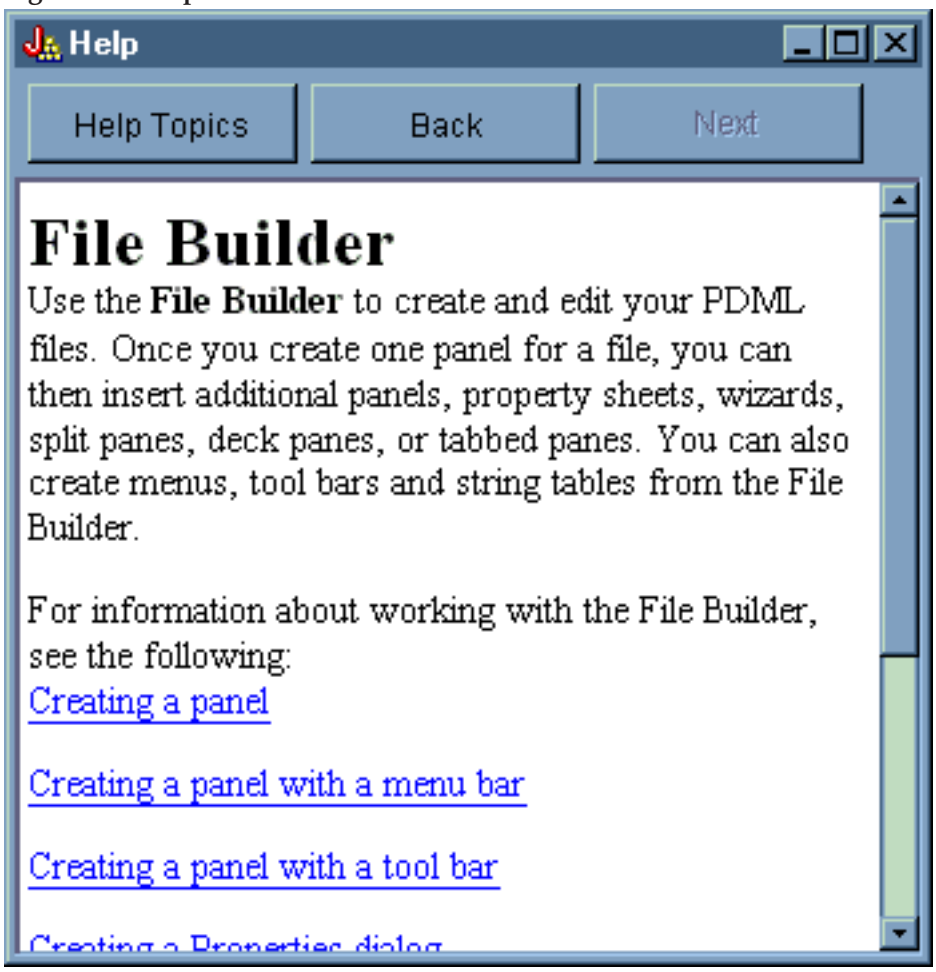
E' possibile suddividere il documento di aiuto dalla riga comandi con il comando che segue:

```
jre com.ibm.as400.ui.tools.HelpDocSplitter "helpdocument.htm" [output directory]
```

Questo comando esegue l'elaborazione che divide il file. E' possibile fornire il nome del Documento di aiuto come immissione con un indirizzario di emissione facoltativo. Per impostazione predefinita, viene creato un sottoindirizzario con lo stesso nome del Documento di aiuto e i file risultanti vengo immessi nell'indirizzario.

Questo è un esempio di come viene visualizzato un file di aiuto:

Figura 3: esempio di file di aiuto GUI Builder

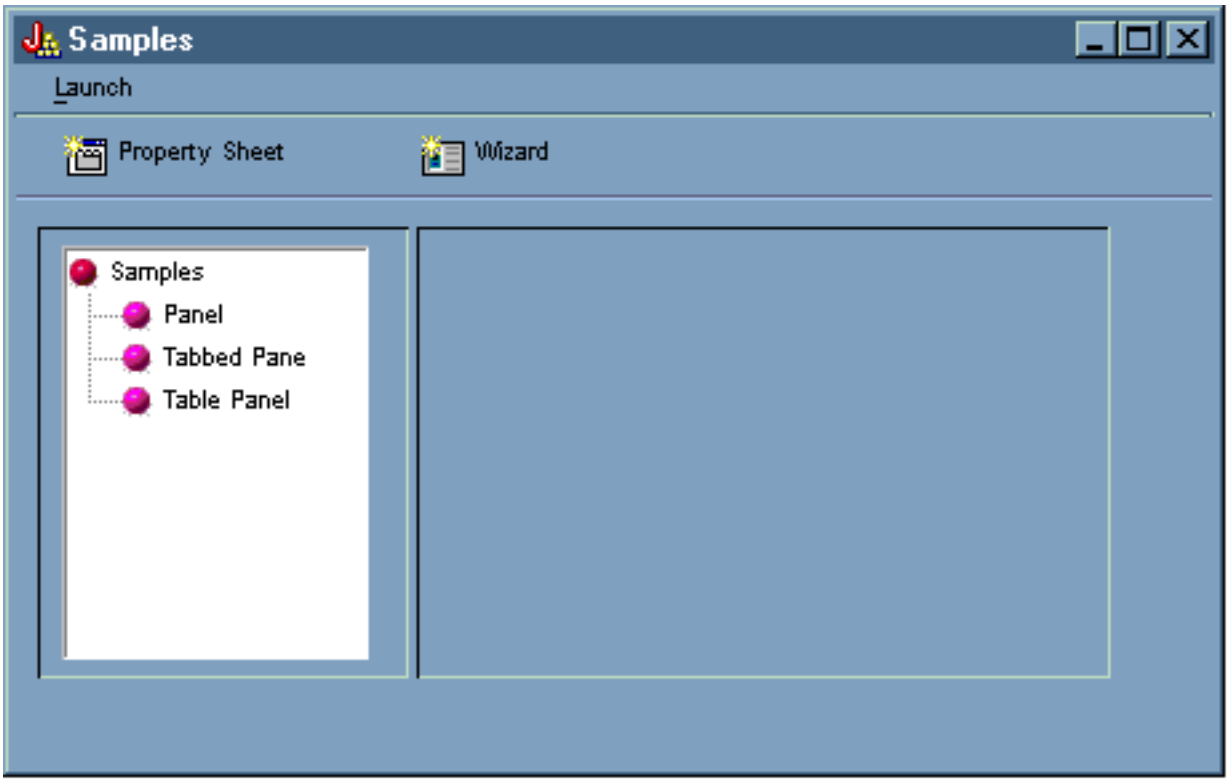


Esempio: utilizzo del GUI Builder

Quando gli esempi contenuti in questa sezione vengono uniti ai bean di dati corretti che operano in secondo piano, si ottiene un'applicazione GUI totale.

La figura 1 mostra il primo pannello che viene visualizzato quando si esegue questo esempio.

Figura 1: finestra principale dell'esempio del GUI Builder



Tenere presente che questo schermo consente di utilizzare il gestore pannelli dinamico.Le figure 2 e 3 mostrano come è possibile ridimensionare la finestra in modo che sia più grande o più piccola.

Figura 2: ridimensionamento della finestra principale dell'esempio del GUI Builder (più grande)

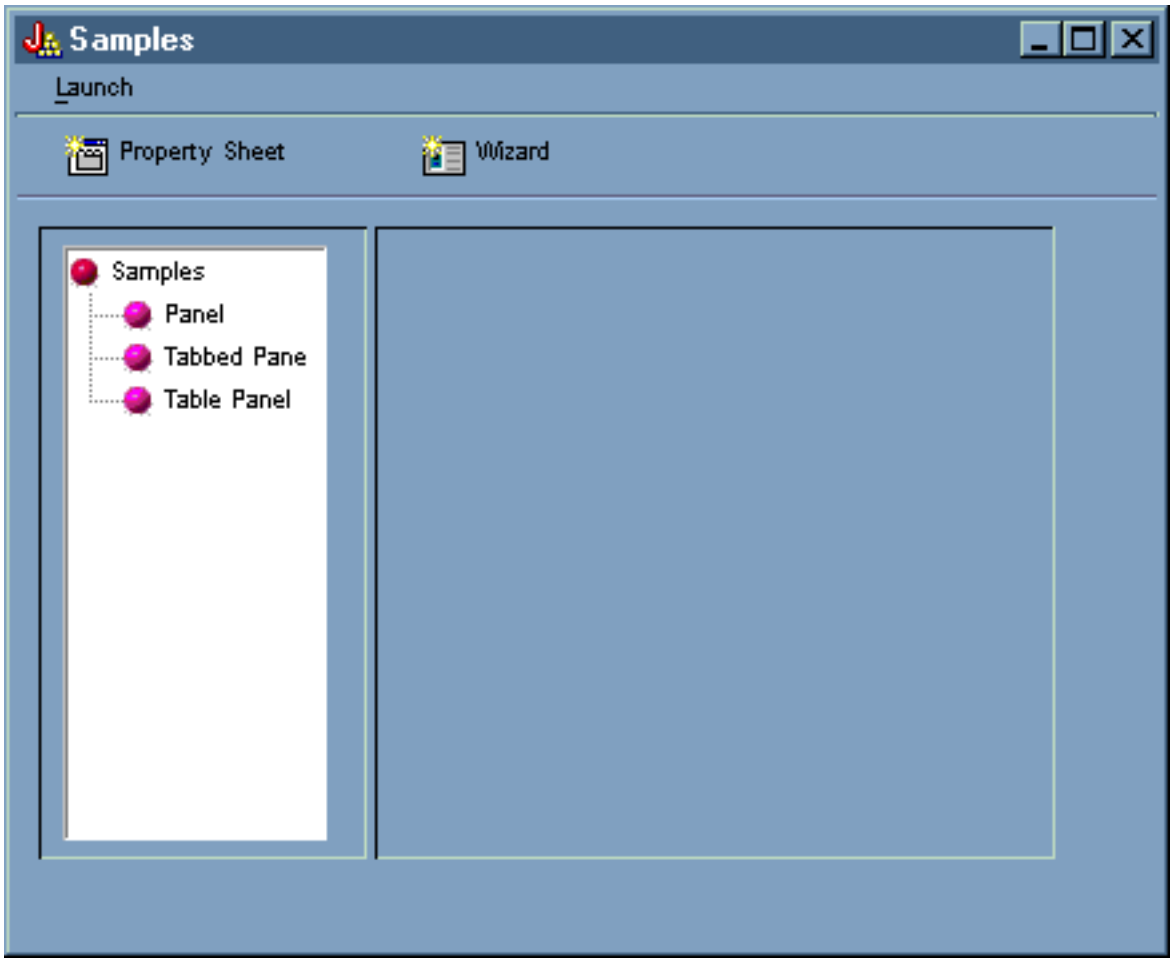
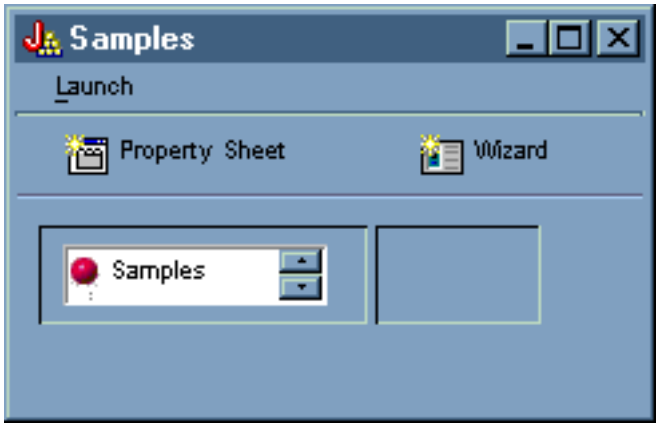


Figura 3: ridimensionamento della finestra principale dell'esempio del GUI Builder (più piccola)



Quando si utilizza il Gestore pannelli dinamico, mentre la dimensione del pannello e i controlli del pannello vengono modificati, la dimensione del testo non subisce cambiamenti.

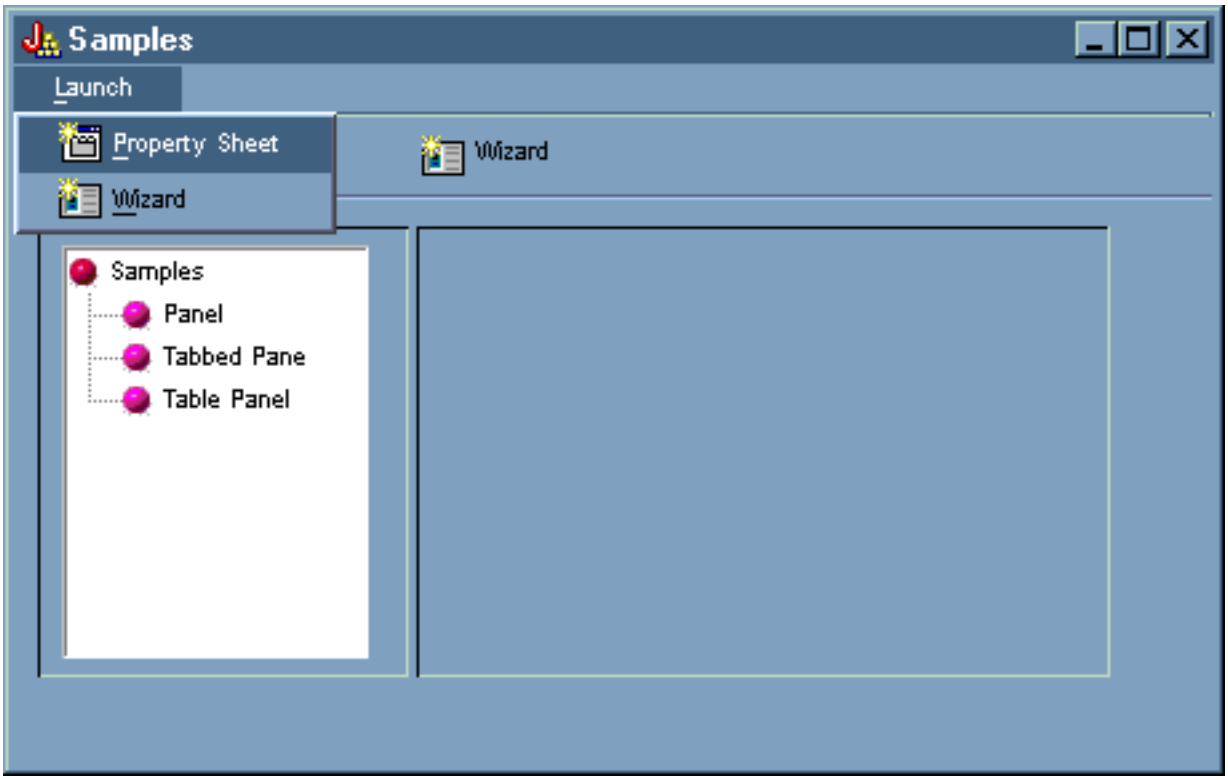
Il pannello consente di svolgere le seguenti azioni:

- Avviare un foglio delle proprietà
- Avviare un wizard
- Visualizzare i campi elencati nel pannello di sinistra

Avvio del foglio delle proprietà

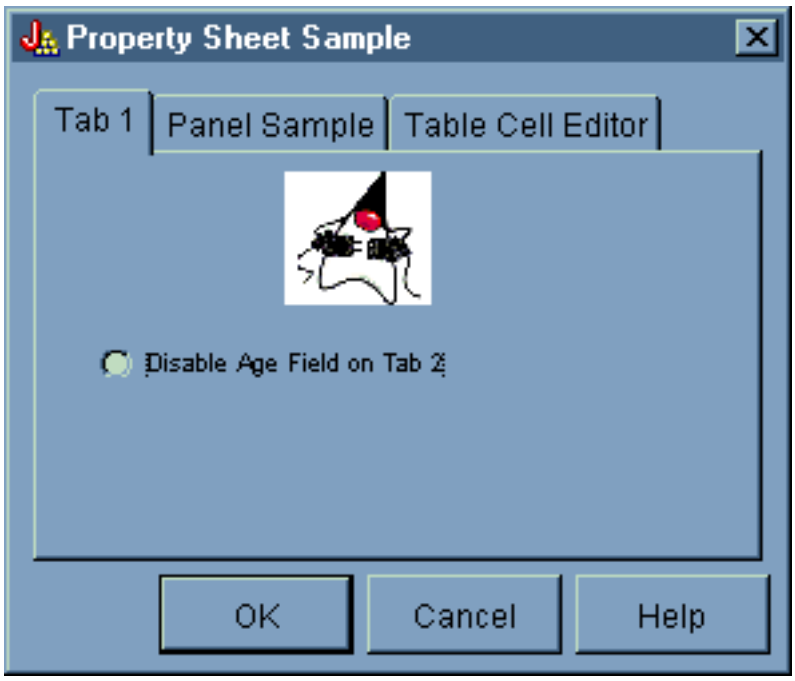
E' possibile avviare il foglio delle proprietà facendo clic sul pulsante della barra degli strumenti Foglio delle proprietà o utilizzando il menu **Avvia**. La possibilità di scegliere tra la barra degli strumenti e il menu consente di visualizzare collegamenti alle voci di menu. La figura 4 mostra il **Foglio delle proprietà** selezionato dal menu **Avvia** sulla finestra principale dell'esempio del GUI Builder.

Figura 4: selezione del Foglio delle proprietà dal menu Avvia



La selezione del **Foglio delle proprietà** visualizza il pannello nella figura 5.

Figura 5: finestra di dialogo Esempio foglio delle proprietà



Quando appare per la prima volta l'Esempio foglio delle proprietà, il separatore 1 viene visualizzato per impostazione predefinita. Le figure 6 e 7 mostrano come la visualizzazione del pannello si modifichi quando si selezionano altri separatori.

Figura 6: selezione del separatore Esempio pannello

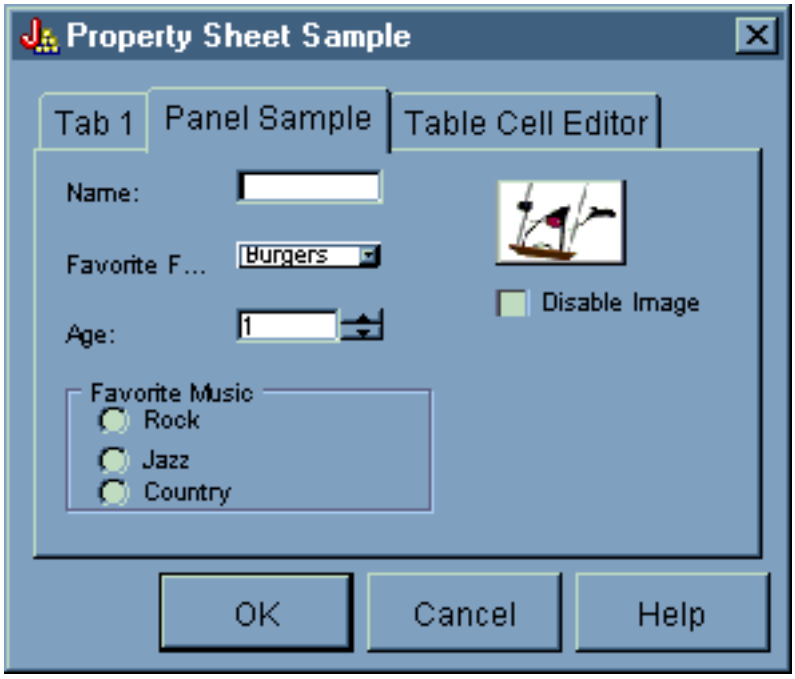
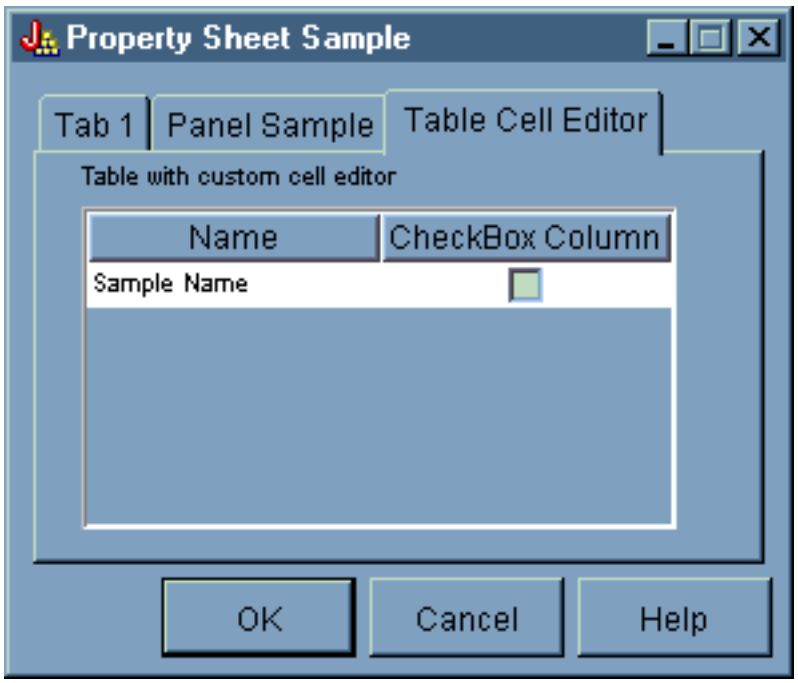


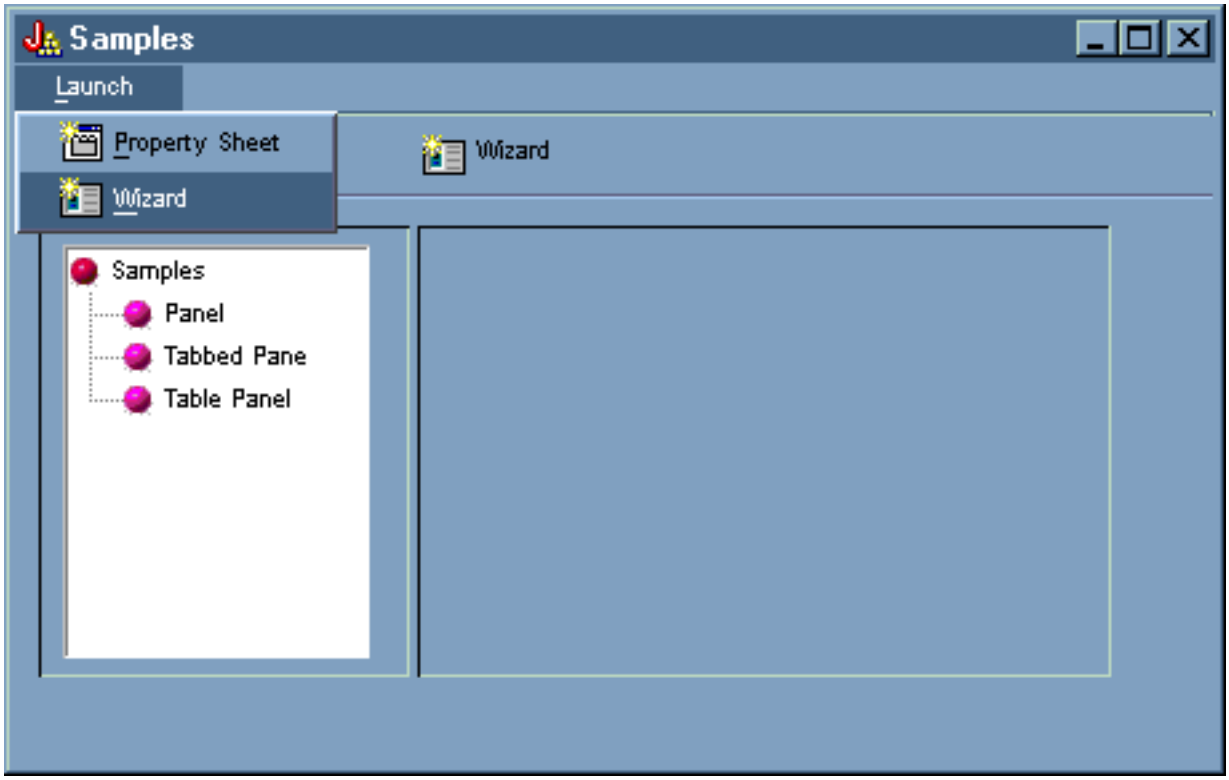
Figura 7: selezione del separatore Editor cella tabella



Avvio del wizard

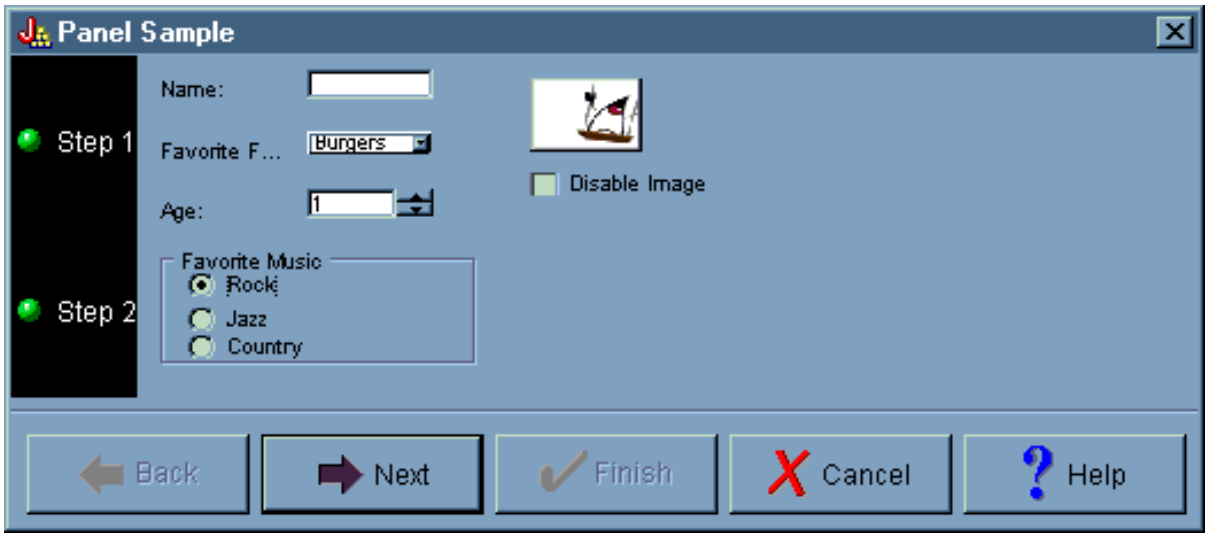
E' possibile avviare il wizard facendo clic sul pulsante Wizard della barra degli strumenti o utilizzando il menu **Avvia**. La possibilità di scegliere tra la barra degli strumenti e il menu consente di visualizzare collegamenti alle voci di menu. La figura 8 mostra il **Wizard** selezionato dal menu **Avvia** nella finestra principale dell'esempio del GUI Builder.

Figura 8: selezione del Wizard dal menu Avvia



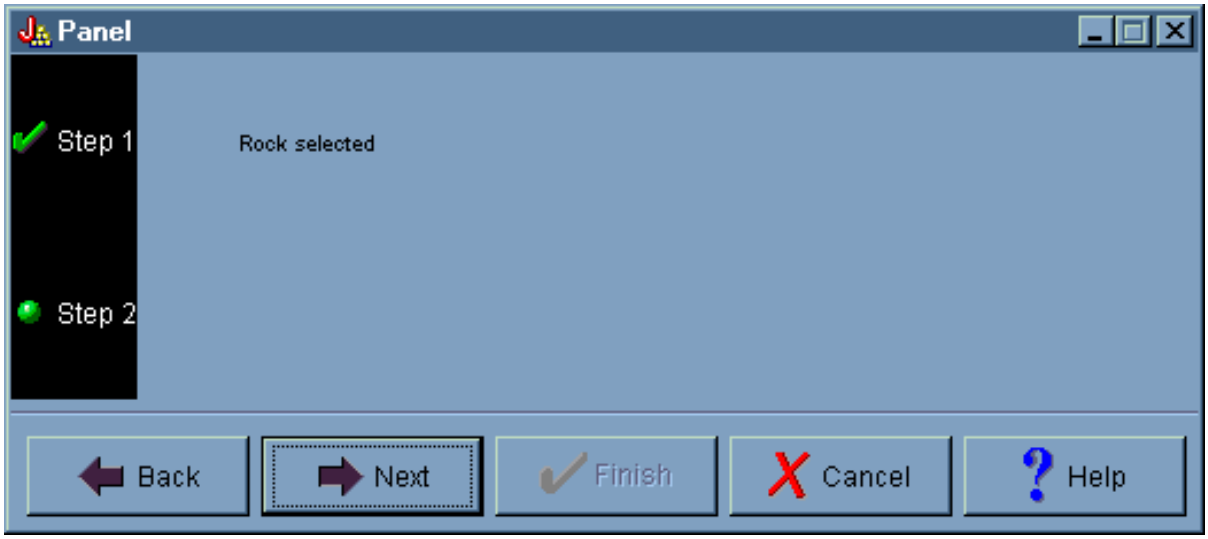
La figura 9 mostra come la prima finestra di dialogo del wizard fornisca molte opzioni.

Figura 9: selezione di Rock nella prima finestra di dialogo del Wizard



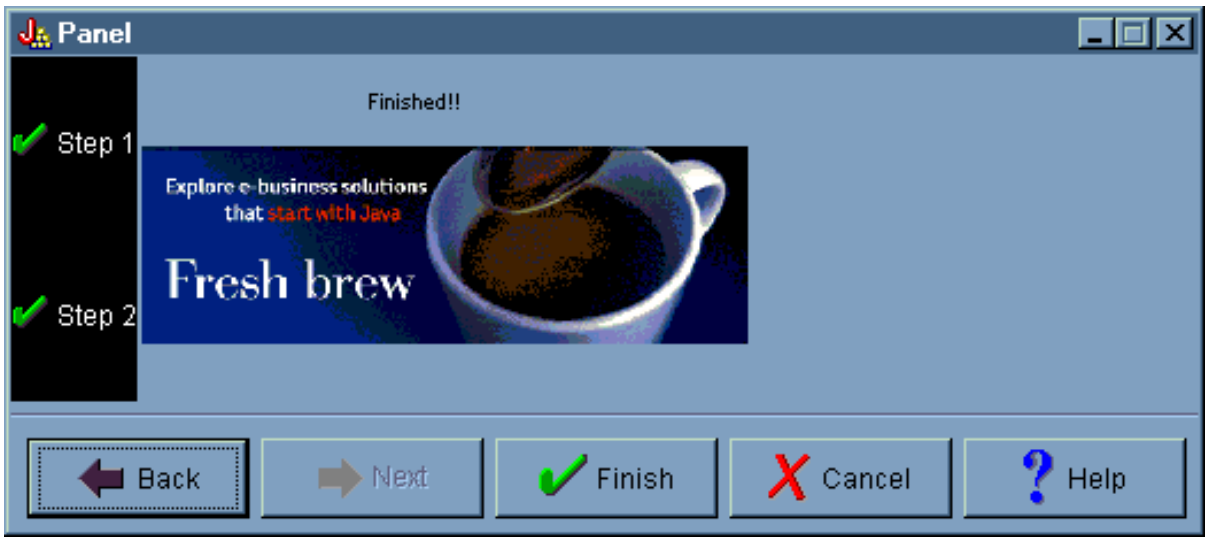
Nella prima finestra di dialogo del wizard, selezionare **Rock** e fare clic su **Avanti** per visualizzare la seconda finestra di dialogo del wizard come mostrato nella figura 10.

Figura 10: la seconda finestra di dialogo del wizard (dopo aver selezionato Rock)



Nella seconda finestra di dialogo del wizard, selezionare **Avanti** per visualizzare la finestra di dialogo finale del wizard come mostrato nella figura 11.

Figura 11: finestra di dialogo finale del wizard



Tuttavia, questo esempio è stato programmato per un loop. Selezionare **Country** nella prima finestra di dialogo del wizard (figura 12), quindi fare clic su **Avanti** per visualizzare la seconda finestra di dialogo del wizard (Figura 13). Facendo clic su Avanti nella seconda finestra di dialogo si esegue il loop per visualizzare di nuovo la prima finestra di dialogo (Figura 14) invece dell'ultima finestra di dialogo del wizard.

Figura 12: selezione di Country nella prima finestra di dialogo del wizard

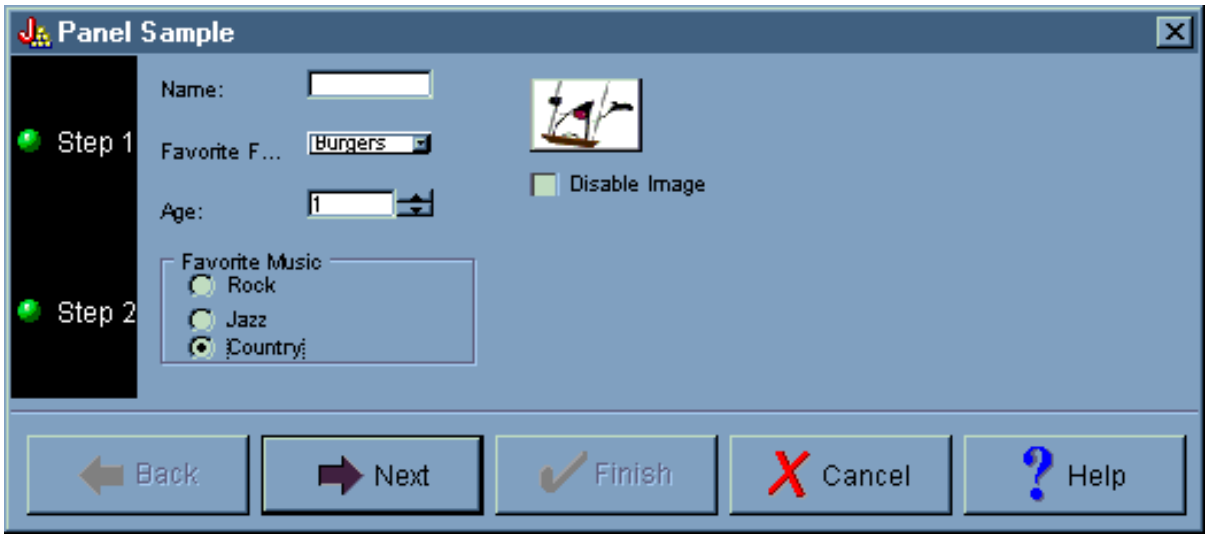


Figura 13: seconda finestra di dialogo del wizard (dopo aver selezionato Country)

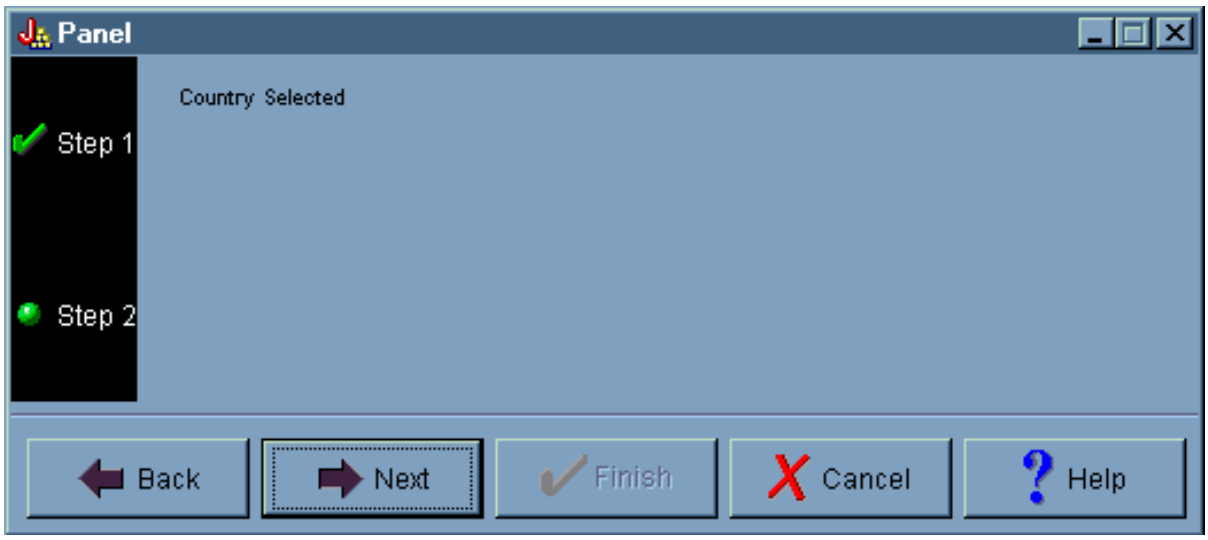
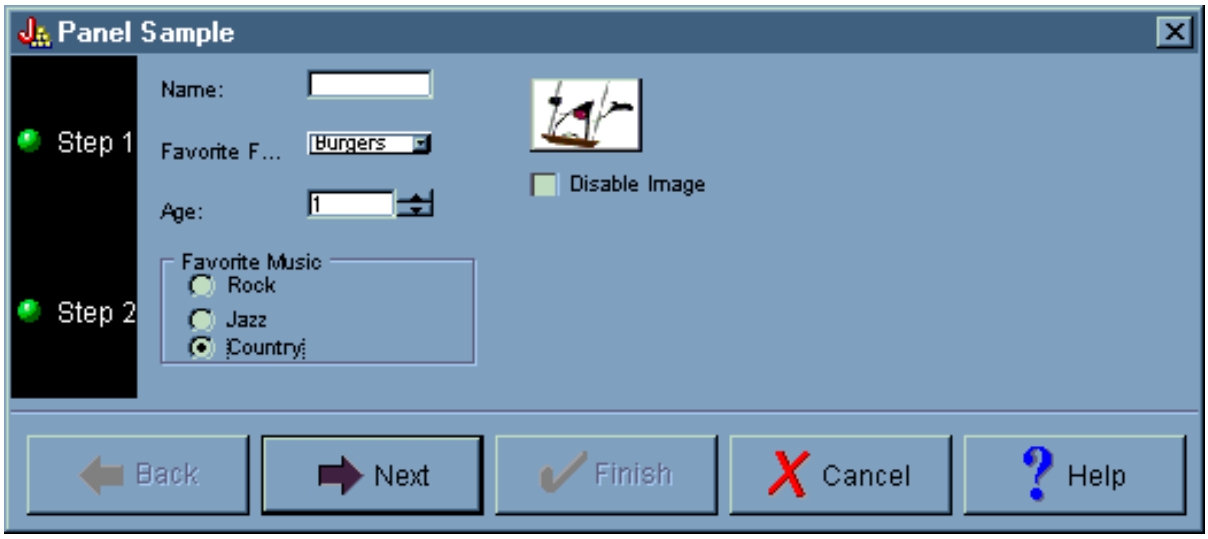


Figura 14: loop alla prima finestra di dialogo del wizard

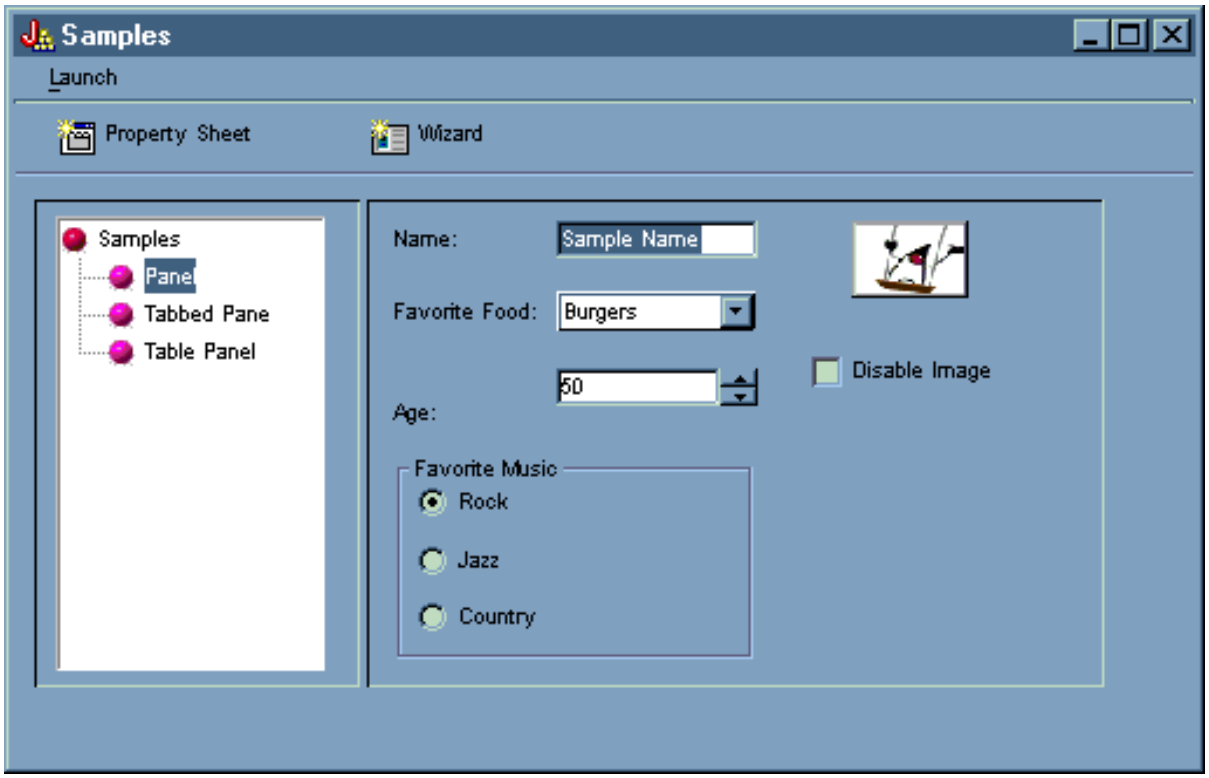


In altre parole, il programmatore ha stabilito che nessuno possa selezionare Country come genere di musica preferito.

Visualizzazione degli esempi

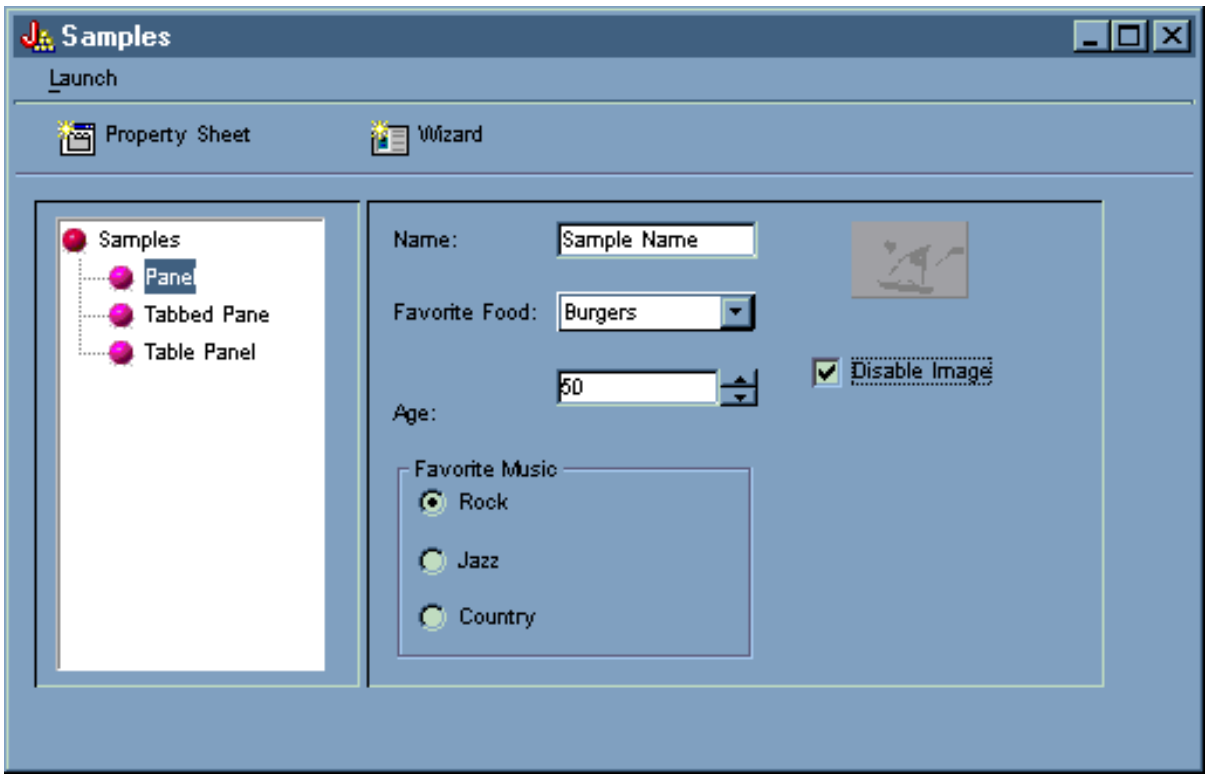
Dalla finestra principale dell'esempio del GUI Builder, è inoltre possibile selezionare altre funzioni dal pannello di sinistra sotto la barra degli strumenti. La figura 15 mostra come la selezione **Pannello** nel pannello di sinistra che visualizza il Campione di pannello nel pannello di destra.

Figura 15: selezione di Pannello nel pannello di sinistra



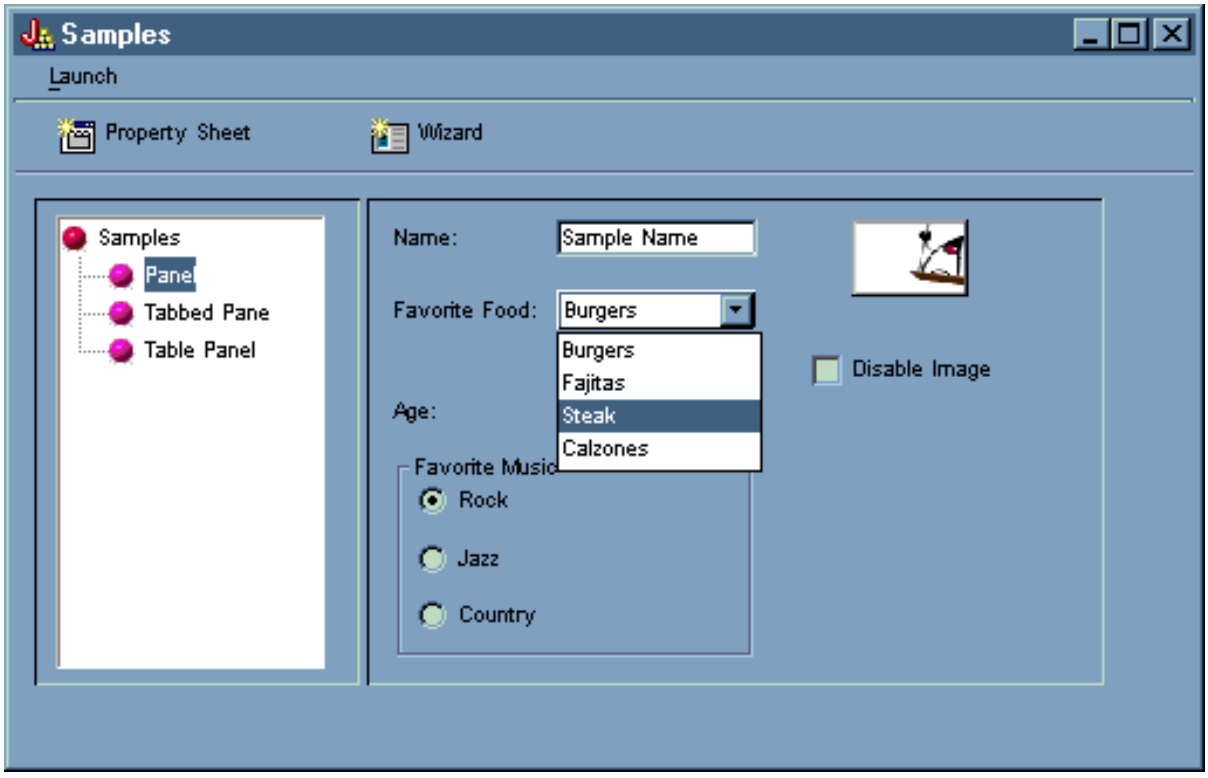
L'Esempio di pannello è stato programmato con un'opzione per disabilitare l'immagine. Selezionare **Disabilita immagine** per visualizzare lo stesso schermo con immagine ingrigita, come mostrato nella Figura 16.

Figura 16: selezione di Disabilita immagine nel pannello di destra



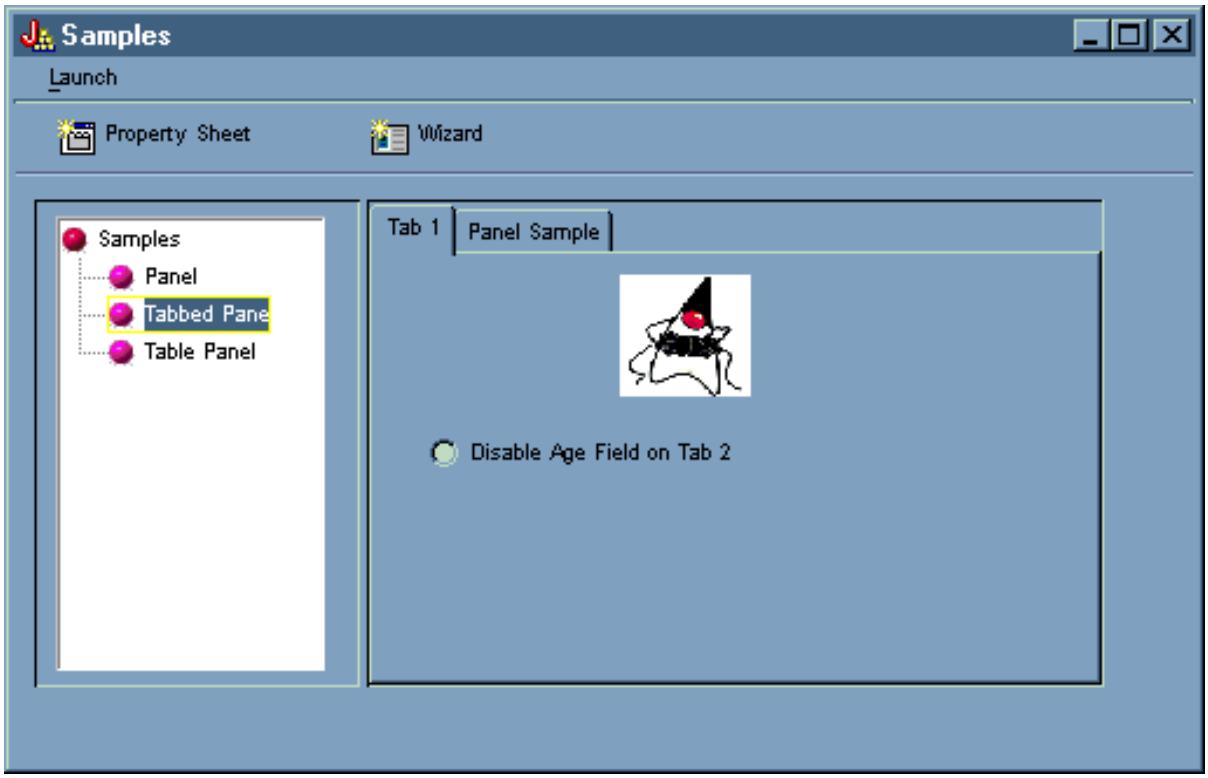
L’esempio di pannello illustra inoltre l’opzione della casella di elenco a discesa, come mostrato nella figura 17.

Figura 17: selezione di una voce dall’elenco Cibi preferiti nel pannello di destra



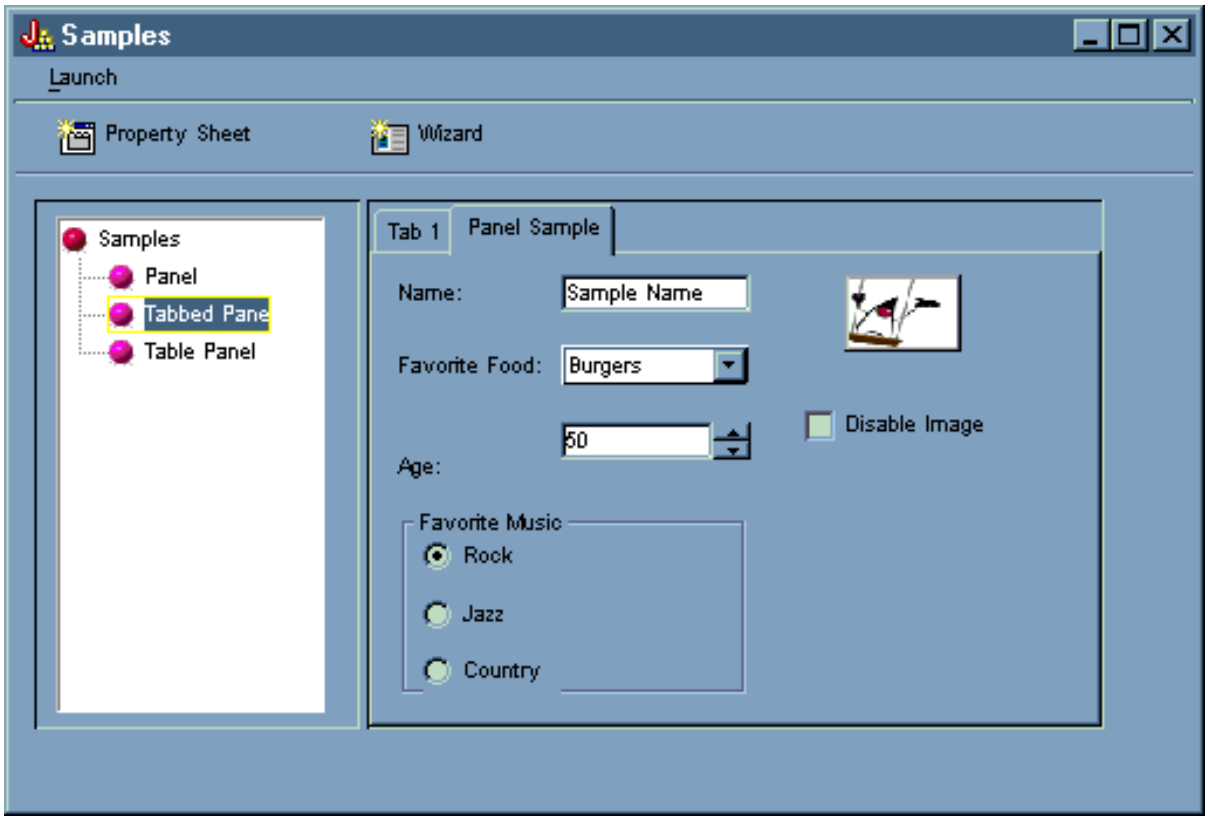
La figura 18 mostra come selezionando il **Pannello con separatori** nel pannello di sinistra della finestra principale dell’esempio del GUI Builder si visualizza il campione di Pannello con separatori nel pannello di destra.

Figura 18: selezione del Pannello con separatori nel pannello di sinistra



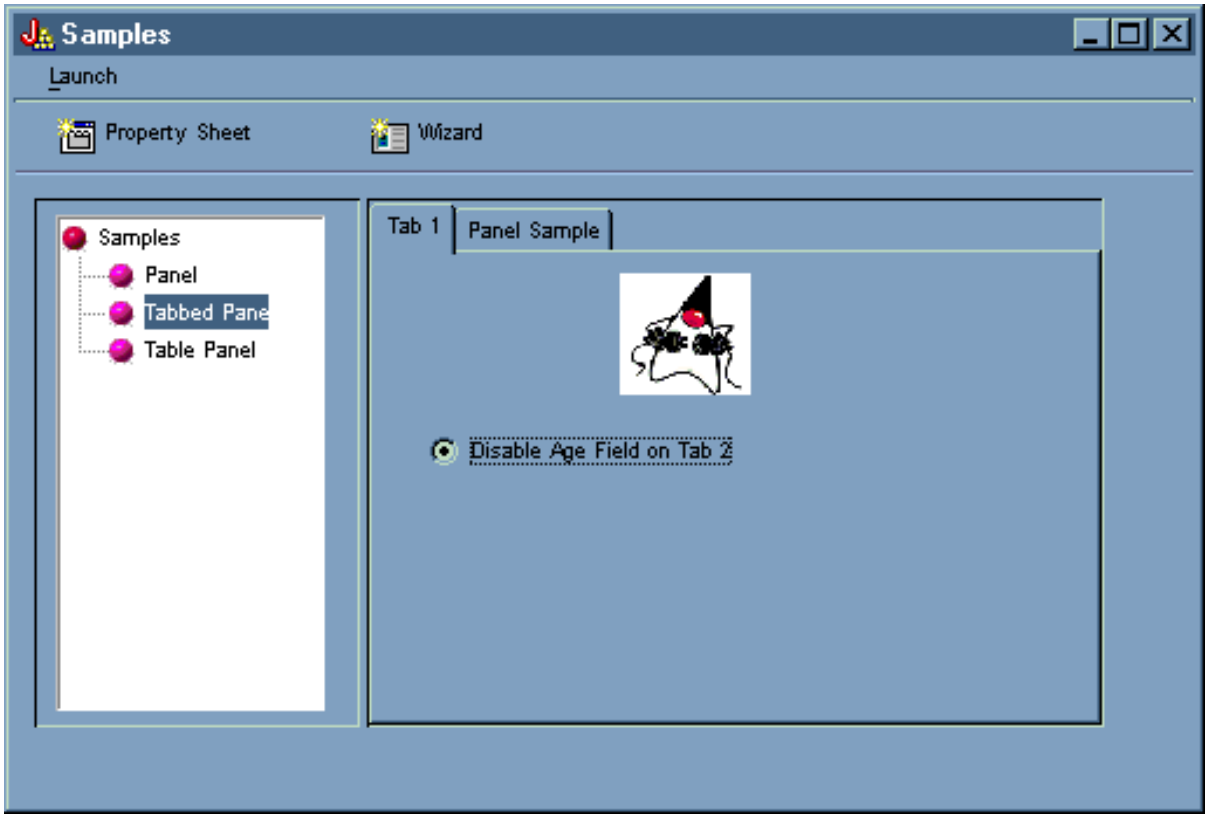
La figura 19 mostra i risultati della selezione del separatore **Esempio pannello** nel pannello di destra.

Figura 19: selezione del separatore Esempio pannello nel pannello di destra



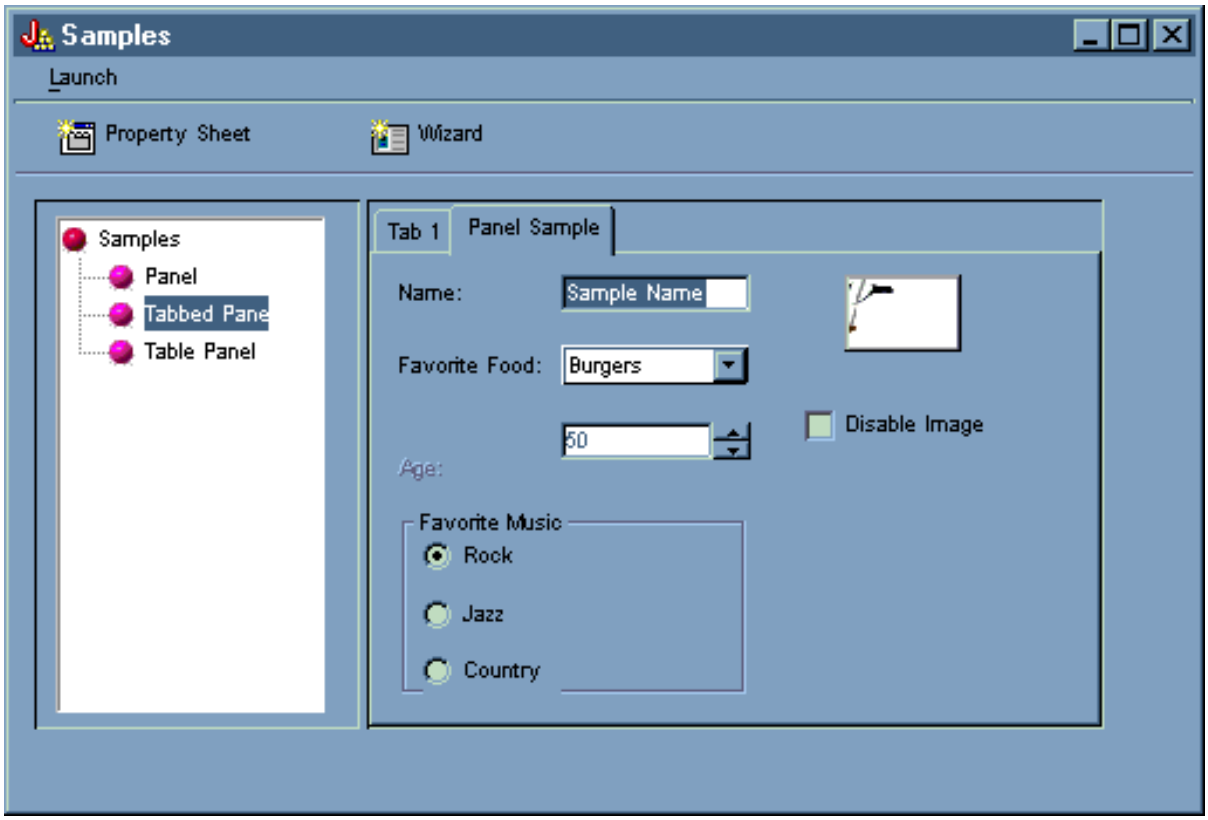
Selezionare di nuovo **Separatore 1** (nel pannello di destra), quindi fare clic su **Disabilita campo età sul Separatore 2** per disabilitarlo.

Figura 20: selezione di Disabilita campo età sul Separatore 2 nel pannello di destra



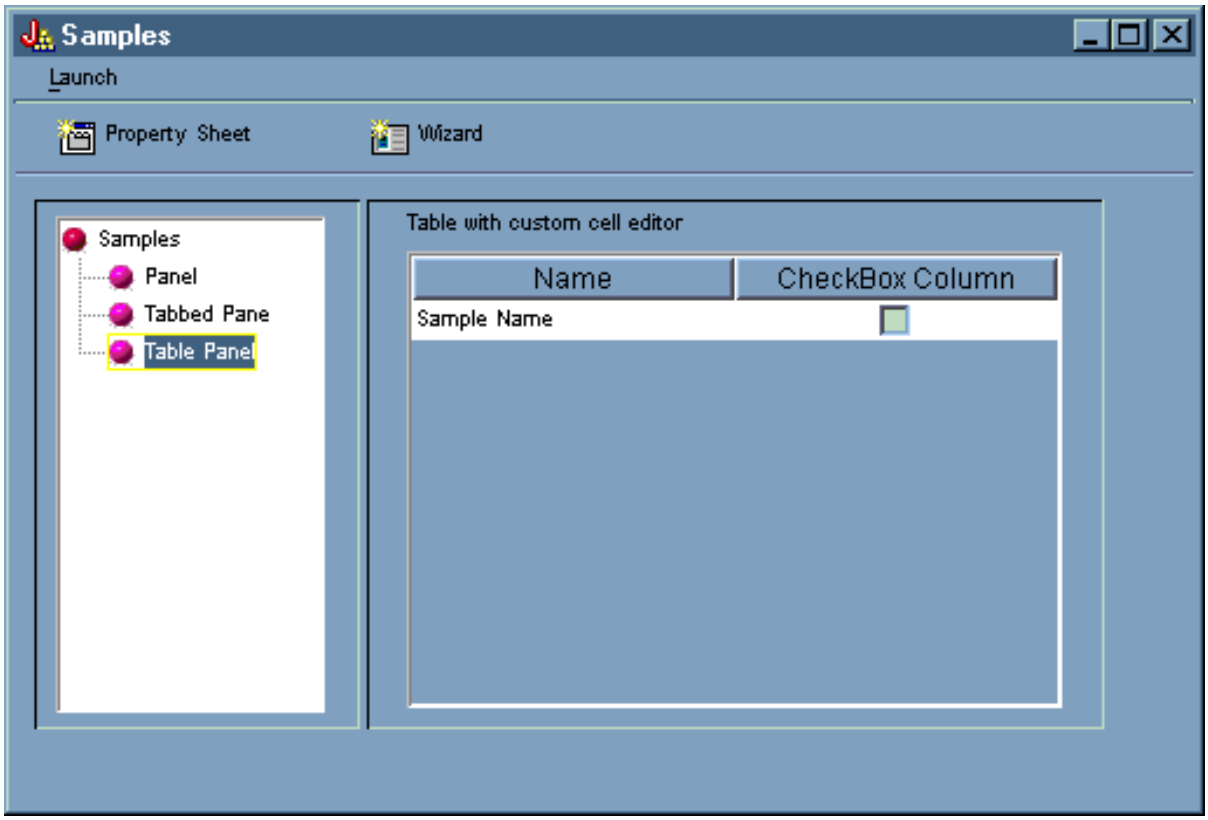
La selezione dell'opzione **Disabilita campo età sul Separatore 2** disattiva e ingrigisce il campo **Età** nel separatore **Esempio pannello**, come mostrato nella figura 21.

Figura 21: risultato della disabilitazione dell'età nel separatore Esempio pannello



La selezione di **Pannello della tabella** nel pannello di sinistra della finestra principale dell'esempio del GUI Builder illustra l'utilizzo di un pannello tabella con un renderer di personalizzazione e un editor cella di personalizzazione, come mostrato nella figura 22.

Figura 22: selezione del Pannello della tabella nel pannello di sinistra



Esempi dalle classi HTML

Gli esempi che seguono mostrano alcuni metodi necessari per utilizzare le classi HTML:

- Esempio: utilizzo della classe BidiOrdering
- Esempio: creazione di oggetti HTMLAlign
- Esempi di classe HTMLDocument:
  - Esempio: utilizzo di HTMLDocument per creare dati HTML
  - Esempio: utilizzo di HTMLDocument per creare dati FO XSL
- Esempio: utilizzo delle classi HTMLForm
- Esempi di classe di immissione modulo:
  - Esempio: creazione di un oggetto ButtonFormInput
  - Esempio: creazione di un oggetto FileFormInput
  - Esempio: creazione di un oggetto HiddenFormInput
  - Esempio: creazione di un oggetto ImageFormInput
  - Esempio: creazione di un oggetto ResetFormInput
  - Esempio: creazione di un oggetto SubmitFormInput
  - Esempio: creazione di un oggetto TextFormInput
  - Esempio: creazione di un oggetto PasswordFormInput
  - Esempio: creazione di un oggetto RadioFormInput
  - Esempio: creazione di un oggetto CheckboxFormInput
- Esempio: creazione di oggetti HTMLHeading
- Esempio: utilizzo della classe HTMLHyperlink
- Esempio: utilizzo della classe HTMLImage
- Esempi HTMLList

  - Esempio: creazione di elenchi ordinati
  - Esempio: creazione di elenchi non ordinati
  - Esempio: creazione di elenchi nidificati
- Esempio: creazione di tag HTMLMeta
- Esempio: creazione di tag HTMLParameter
- Esempio: creazione di tag HTMLServlet
- Esempio: utilizzo della classe HTMLText
- Esempi HTMLTree
  - Esempio: utilizzo della classe HTMLTree
  - Esempio: creazione di una gerarchia ad albero IFS
- Classi LayoutForm:
  - Esempio: utilizzo della classe GridLayoutFormPanel
  - Esempio: utilizzo della classe LineLayoutFormPanel
- Esempio: utilizzo della classe TextAreaFormElement
- Esempio: utilizzo della classe LabelFormOutput
- Esempio: utilizzo della classe SelectFormElement
- Esempio: utilizzo della classe SelectOption
- Esempio: utilizzo della classe RadioFormInputGroup
- Esempio: utilizzo della classe RadioFormInput
- Esempio: utilizzo delle classi HTMLTable
  - Esempio: utilizzo della classe HTMLTableCell
  - Esempio: utilizzo della classe HTMLTableRow
  - Esempio: utilizzo della classe HTMLTableHeader
  - Esempio: utilizzo della classe HTMLTableCaption

E' inoltre possibile utilizzare le classi HTML e servlet insieme, come in questo esempio.

Il seguente esonero di responsabilità si applica a tutti gli esempi IBM Toolbox per Java:

> **Esonero di responsabilità per gli esempi di codice**
>
> L'IBM fornisce una licenza non esclusiva per utilizzare tutti gli esempi del codice di programmazione da cui creare funzioni simili personalizzate, in base a richieste specifiche.
>
> Questo codice di esempio è fornito dall'IBM con la sola funzione illustrativa. Questi esempi non sono stati interamente testati in tutte le condizioni. IBM, perciò, non fornisce nessun tipo di garanzia o affidabilità implicita, rispetto alla funzionalità o alle funzioni di questi programmi.
>
> Tutti i programmi qui contenuti vengono forniti all'utente ″COSI' COME SONO″ senza garanzie di alcun tipo. Le garanzie implicite di non contraffazione, commerciabilità e adeguatezza a scopi specifici sono espressamente vietate.

Esempio: utilizzo delle classi modulo HTML

L'esempio che segue mostra come utilizzare le classi modulo HTML. E' inoltre possibile visualizzare un'emissione campione ricavata dall'esecuzione di questo codice. Le classi HTML utilizzate nel metodo ″showHTML″ sono in **grassetto**.

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per informazioni legali importanti.

```
///////////////////////////////////////////////////////////////////////////
//
// Questo sorgente è un esempio di utilizzo delle classi pacchetto HTML di IBM Toolbox,
// per Java, che consentono di creare facilmente Moduli HTML.
//
///////////////////////////////////////////////////////////////////////////

package customer;

import java.io.*;
import java.awt.Color;

import javax.servlet.*;
import javax.servlet.http.*;

import com.ibm.as400.access.*;
import com.ibm.as400.util.html.*;


public class HTMLExample extends HttpServlet
{
  // Stabilisce se l'utente esiste già nell'elenco di utenti registrati.
  private static boolean found = false;

  // Le informazioni sulla registrazione verranno memorizzate qui
  String regPath = "c:\\registration.txt";


     public void init(ServletConfig config)
  {

                                try
    {
             super.init(config);

    }
      catch(Exception e)
    {
               e.printStackTrace();
    }

  }

  /**
       *  Process the GET request.
       *  @param req The request.
       *  @param res The response.
   **/
      public void doGet (HttpServletRequest req, HttpServletResponse res)
            throws ServletException, IOException
   {
            res.setContentType("text/html");
            ServletOutputStream out = res.getOutputStream();

            // Visualizzare il Web tramite le nuove classi HTML
            out.println(showHTML());
            out.close();
   }


       public void doPost (HttpServletRequest req, HttpServletResponse res)
             throws ServletException, IOException
    {
                String nameStr   = req.getParameter("name");
                String emailStr = req.getParameter("email");
                String errorText= "";
```

```
        // Flusso di emissione da scrivere nel servlet
      ServletOutputStream out = res.getOutputStream();

      res.setContentType("text/html");

        // Controllare che i parametri name & e-mail abbiano valori validi
        if (nameStr.length() == 0)
   errorText += "Customer Name not entered.  ";
        if (emailStr.length() == 0)
   errorText += "E-mail not entered.  " ;

        // Se sono stati forniti name & e-mail, continuare.
        if (errorText.length() == 0)
{

                              try
   {
                  //Creare il file registration.txt
                  FileWriter f = new FileWriter(regPath, true);
                  BufferedWriter output = new BufferedWriter(f);

                  //il programma di lettura memorizzato nel buffer per la ricerca del file
                  BufferedReader in = new BufferedReader(new FileReader(regPath));

                  String line = in.readLine();

                  // reimpostato l'indicatore found
                  found = false;

                  // Controllare per verificare se questo cliente è già stato registrato
                  //  o ha già utilizzato lo stesso indirizzo e-mail
                  while (!found)
      {
                        // se il file è vuoto o è stata raggiunta l'EOF.
                        if (line == null)
                                  break;

                        // se il cliente è già stato registrato
         if ((line.equals("Customer Name: " + nameStr)) ||
             (line.equals("Email address: " + emailStr)))
         {
            // Inviare un messaggio al cliente informandolo che è già
            // registrato
            out.println ("<HTML> " +
                        "<TITLE> Toolbox Registration</TITLE> " +
                        "<META HTTP-EQUIV=\"pragma\" content=\"no-cache\"> " +
                        "<BODY BGCOLOR=\"blanchedalmond\" TEXT=\"black\"> " );
            out.println ("<P><HR>" +
                        "<P>" + nameStr +
                        "</B>, you have already registered using that " +
                        "<B>Name</B> or <B>E-mail address</B>." +
                        "<P> Thank You!...<P><HR>");

                       // Creare un oggetto HTMLHyperlink e visualizzarlo
            out.println ("<UL><LI>" +
                        new HTMLHyperlink("./customer.HTMLExample",
                                          "Back to Registration Form") +
                        "</UL></BODY></HTML>");
                               found = true;
                                   break;

         }
                         else    // leggere la riga successiva
                               line = in.readLine();

      }
```

```
              // Oggetto stringa per contenere i dati inoltrati dal Modulo HTML
              String data;

// Se il nome degli utenti o l'e-mail non vengono rilevati
// nel file di testo, continuare.
              if (!found)
{
   //-------------------------------------------------------------
                  // Inserire le info sul nuovo cliente in un file
                        output.newLine();
                  output.write("Customer Name: " + nameStr);
                        output.newLine();
                  output.write("Email address: " + emailStr);
                        output.newLine();
   //-------------------------------------------------------------


   //-------------------------------------------------------------
                  //Richiamo della casella di spunta "USE" dal modulo
                  data = req.getParameter("use");
                  if(data != null)
   {
                        output.write("Currently Using Toolbox: " + data);
                           output.newLine();
   }
   //-------------------------------------------------------------


   //-------------------------------------------------------------
                  //Richiamo della casella di spunta "Ulteriori informazioni" dal modulo
                  data = req.getParameter("contact");
           if (data != null)
   {
                        output.write("Requested More Information: " + data);
                           output.newLine();
   }
   //-------------------------------------------------------------


   //-------------------------------------------------------------
                  //Richiamo di "Versione AS400" dal modulo
                  data = req.getParameter("version");
           if (data != null)
   {
                        if (data.equals("multiple versions"))
      {
                              data = req.getParameter("MultiList");
                              output.write("Multiple Versions: " + data);
      }
                             else
                              output.write("AS400 Version: " + data);

                           output.newLine();
   }
   //-------------------------------------------------------------


   //-------------------------------------------------------------
                  //Richiamo di "Progetti correnti" dal modulo
                  data = req.getParameter("interest");
           if (data != null)
   {
                        output.write("Using Java or Interested In: " + data);
                           output.newLine();
   }
   //-------------------------------------------------------------
```

```
//-------------------------------------------------------------
                //Richiamo di "Piattaforme" dal modulo
                data = req.getParameter("platform");
        if (data != null)
{
                      output.write("Platforms: " + data);
                         output.newLine();
   if (data.indexOf("Other") >= 0)
   {
      output.write("Other Platforms: " + req.getParameter("OtherPlatforms"));
      output.newLine();
   }
}
//-------------------------------------------------------------


//-------------------------------------------------------------
//Richiamare "Numero dei server iSeries" dal modulo
                data = req.getParameter("list1");
        if (data != null)
{
                      output.write("Number of iSeries servers: " + data);
                         output.newLine();
}
//-------------------------------------------------------------


//-------------------------------------------------------------
                //Richiamo di "Commenti" dal modulo
                data = req.getParameter("comments");
if (data != null && data.length() > 0)
{
                      output.write("Comments: " + data);
                         output.newLine();
}
//-------------------------------------------------------------


//-------------------------------------------------------------
                //Richiamo di "Allegato"
                data = req.getParameter("myAttachment");
if (data != null && data.length() > 0)
{
                      output.write("Attachment File: " + data);
                         output.newLine();
}
//-------------------------------------------------------------


//-------------------------------------------------------------
                //Richiamo delle informazioni sul "Copyright" nascoste
                data = req.getParameter("copyright");
        if (data != null)
{
                      output.write(data);
                         output.newLine();
}
//-------------------------------------------------------------


                output.flush();
                output.close();

                // Stampare un ringraziamento al cliente
out.println("<HTML>");
out.println("<TITLE>Thank You!</TITLE>");
```

```
            out.println("<META HTTP-EQUIV=\"pragma\" content=\"no-cache\"> ");
            out.println("<BODY BGCOLOR=\"blanchedalmond\">");
            out.println("<HR><P>Thank You for Registering, <B>" + nameStr + "</B>!<P><HR>");

                        // Creare un oggetto HTMLHyperlink e visualizzarlo
            out.println("<UL><LI>" +
                         new HTMLHyperlink("./customer.HTMLExample", "Back to Registration Form"));
            out.println("</UL></BODY></HTML>");

          }

        }
                                  catch (Exception e)
       {
                        // Visualizzare l'errore nel browser
          out.println("<HTML>");
          out.println("<TITLE>ERROR!</TITLE>");
          out.println("<META HTTP-EQUIV=\"pragma\" content=\"no-cache\"> ");
          out.println("<BODY BGCOLOR=\"blanchedalmond\">");
          out.println("<BR><B>Error Message:</B><P>");
          out.println(e + "<P>");

                     // Creare un oggetto HTMLHyperlink e visualizzarlo
          out.println("<UL><LI>" +
                      new HTMLHyperlink("./customer.HTMLExample", "Back to Registration Form"));
          out.println("</UL></BODY></HTML>");

                  e.printStackTrace();
       }
    }
                           else
    {
       // Inviare un messaggio al client informandolo che il nome &
       // e l'e-mail non sono stati immessi. E' necessario ripetere l'operazione
       out.println ("<HTML> " +
                    "<TITLE>Invalid Registration Form</TITLE> " +
                    "<META HTTP-EQUIV=\"pragma\" content=\"no-cache\"> " +
                    "<BODY BGCOLOR=\"blanchedalmond\" TEXT=\"black\"> " );

       out.println ("<HR><B>ERROR</B> in customer data - <P><B>" +
                    errorText +
                    "</B><P>Please Try Again... <HR>");


                   // Creare un oggetto HTMLHyperlink e visualizzarlo
       out.println("<UL><LI>" +
                    new HTMLHyperlink("./customer.HTMLExample", "Back to Registration Form") +
                    "</UL></BODY></HTML>");
    }
             // Chiudere il programma di scrittura
           out.close();


}


    public void destroy(ServletConfig config)
{
         // nessuna operazione
}


    public String getServletInfo()
{
            return "My Product Registration";
}
```

```
   private String showHTML()
{
       // Buffer di stringa per contenere la Pagina HTML
     StringBuffer page = new StringBuffer();

       // Creare l'oggetto modulo HTML
  HTMLForm form = new HTMLForm("/servlet/customer.HTMLExample");;
       HTMLText txt;

       // Creare l'inizio della Pagina HTML ed aggiungerlo al Buffer di stringa
  page.append("<HTML>\n");
  page.append("<TITLE> Welcome!!</TITLE>\n");
  page.append("<HEAD><SCRIPT LANGUAGE=\"JavaScript\">
              function test(){alert(\"This is a sample script executed with a
              ButtonFormInput.\")}</SCRIPT></HEAD>");
  page.append("<META HTTP-EQUIV=\"pragma\" content=\"no-cache\">\n");
  page.append("<BODY BGCOLOR=\"blanchedalmond\" TEXT=\"black\"><BR>\n");

                            try
  {
     //-----------------------------------------------------------------
              // Creare un titolo pagina utilizzando Testo HTML
             txt = new HTMLText("Product Registration");
             txt.setSize(5);
             txt.setBold(true);
              txt.setColor(new Color(199, 21, 133));
              txt.setAlignment(HTMLConstants.CENTER);

              // Aggiungere Testo HTML al Buffer di stringa
     page.append(txt.getTag(true) + "<HR><BR>\n");
     //-----------------------------------------------------------------

     //-----------------------------------------------------------------
              // Creare un Layout di riga
              LineLayoutFormPanel line = new LineLayoutFormPanel();
             txt = new HTMLText("Enter your name and e-mail address:");
              txt.setSize(4);
              line.addElement(txt);

              // Aggiungere il Layout di riga al Buffer di stringa
              page.append(line.toString());
     page.append("<BR>");
     //-----------------------------------------------------------------

     //-----------------------------------------------------------------
              // Impostare METHOD del modulo HTML
             form.setMethod(HTMLForm.METHOD_POST);
     //-----------------------------------------------------------------

     //-----------------------------------------------------------------
              // Creare un'immissione Testo per il nome.
              TextFormInput user = new TextFormInput("name");
              user.setSize(25);
              user.setMaxLength(40);

              // Creare un'immissione Testo per l'indirizzo email.
              TextFormInput email = new TextFormInput("email");
              email.setSize(30);
              email.setMaxLength(40);

              // Creare una ImageFormInput
     ImageFormInput img =
       new ImageFormInput("Submit Form", "..\\images\\myPiimages/c.gif");
              img.setAlignment(HTMLConstants.RIGHT);
     //-----------------------------------------------------------------

     //-----------------------------------------------------------------
```

```
        // Creare un oggetto LineLayoutFormPanel per il nome & l'indirizzo e-mail
        LineLayoutFormPanel line2 = new LineLayoutFormPanel();

        // Aggiungere elementi al formato della riga
        line2.addElement(new LabelFormElement("Name:"));
        line2.addElement(user);
        // Creare ed aggiungere un Elemento etichetta al Layout di riga
        line2.addElement(new LabelFormElement("E-mail:"));
        line2.addElement(email);
        line2.addElement(img);
//-----------------------------------------------------------------------

//-----------------------------------------------------------------------
        // Creare layout della riga di domanda
        LineLayoutFormPanel line3 = new LineLayoutFormPanel();

        // Aggiungere elementi al layout di riga
        line3.addElement(new LineLayoutFormPanel());
line3.addElement(new
  CheckboxFormInput("use",
                    "yes",
                    "Do you currently use the Toolbox?",
                    false));
        line3.addElement(new LineLayoutFormPanel());
line3.addElement(new CheckboxFormInput(
   "contact",
   "yes",
   "Would you like information on future Toolbox releases?",
   true));
        line3.addElement(new LineLayoutFormPanel());
//-----------------------------------------------------------------------

//-----------------------------------------------------------------------
        // Creare gruppo Radio di versione
        RadioFormInputGroup group = new RadioFormInputGroup("version");

        // Aggiungere immissioni modulo Radio al gruppo
        group.add(new RadioFormInput("version", "v3r2", "V3R2", false));
        group.add(new RadioFormInput("version", "v4r1", "V4R1", false));
        group.add(new RadioFormInput("version", "v4r2", "V4R2", false));
        group.add(new RadioFormInput("version", "v4r3", "V4R3", false));
        group.add(new RadioFormInput("version", "v4r4", "V4R4", false));
group.add(new
  RadioFormInput("version",
                 "multiple versions",
                 "Multiple Versions? Which ones:",
                 false));

        //Creare un elemento modulo di selezione
        SelectFormElement mlist = new SelectFormElement("MultiList");
        mlist.setMultiple(true);
        mlist.setSize(3);

        //Creare le opzioni per l'elemento modulo di selezione
        SelectOption option1 = mlist.addOption("V3R2", "v3r2");
        SelectOption option2 = mlist.addOption("V4R1", "v4r1");
        SelectOption option3 = mlist.addOption("V4R2", "v4r2");
        SelectOption option4 = mlist.addOption("V4R3", "v4r3");
        SelectOption option5 = mlist.addOption("V4R4", "v4r4");

        // Creare testo HTML
txt = new HTMLText("Current Server Level:");
        txt.setSize(4);

        // Creare layout di griglia
        GridLayoutFormPanel grid1 = new GridLayoutFormPanel(3);
```

```
        // Aggiungere gruppo radio & elemento modulo di selezione alla griglia
        grid1.addElement(txt);
        grid1.addElement(group);
        grid1.addElement(mlist);
//---------------------------------------------------------------------

//---------------------------------------------------------------------
        // Creare layout di griglia per interessi
        GridLayoutFormPanel grid2 = new GridLayoutFormPanel(1);
txt = new HTMLText("Current Projects or Area of Interest: " +
                   "(check all that apply)");
        txt.setSize(4);

        // Aggiungere elementi al layout di griglia
        grid2.addElement(new LineLayoutFormPanel());
        grid2.addElement(txt);
        // Creare ed aggiungere una casella di spunta al layout di griglia
grid2.addElement(new
  CheckboxFormInput("interest", "applications", "Applications", true));
grid2.addElement(new
  CheckboxFormInput("interest", "applets", "Applets", false));
grid2.addElement(new
  CheckboxFormInput("interest", "servlets", "Servlets", false));
//---------------------------------------------------------------------

//---------------------------------------------------------------------
        // Creare layout di riga per piattaforme
        LineLayoutFormPanel line4 = new LineLayoutFormPanel();
txt = new HTMLText("Client Platforms Used: " +
                   "(check all that apply)");
        txt.setSize(4);

        // Aggiungere elementi al layout di riga
        line4.addElement(new LineLayoutFormPanel());
        line4.addElement(txt);
        line4.addElement(new LineLayoutFormPanel());
line4.addElement(new CheckboxFormInput("platform",
                                       "95",
                                       "Windows95",
                                       false));
line4.addElement(new CheckboxFormInput("platform",
                                       "98",
                                       "Windows98",
                                       false));
line4.addElement(new CheckboxFormInput("platform",
                                       "NT",
                                       "WindowsNT",
                                       false));
line4.addElement(new CheckboxFormInput("platform",
                                       "OS2",
                                       "OS/2",
                                       false));
line4.addElement(new CheckboxFormInput("platform",
                                       "AIX",
                                       "AIX",
                                       false));
line4.addElement(new CheckboxFormInput("platform",
                                       "Linux",
                                       "Linux",
                                       false));
line4.addElement(new CheckboxFormInput("platform",
                                       "AS400",
                                       "iSeries",
                                       false));
line4.addElement(new CheckboxFormInput("platform",
                                       "Other",
                                       "Other:",
```

```
                                        false));

        TextFormInput other = new TextFormInput("OtherPlatforms");
        other.setSize(20);
        other.setMaxLength(50);

        line4.addElement(other);
//-------------------------------------------------------------------

//-------------------------------------------------------------------
        // Creare un layout di riga per numero di server
        LineLayoutFormPanel grid3 = new LineLayoutFormPanel();

txt = new HTMLText(
   "How many iSeries servers do you have?");
        txt.setSize(4);

        // Creare un elemento modulo di selezione per numero di server posseduti
        SelectFormElement list = new SelectFormElement("list1");
        // Creare ed aggiungere Opzioni di selezione all'elenco di elementi modulo di selezione
        SelectOption opt0 = list.addOption("0", "zero");
        SelectOption opt1 = list.addOption("1", "one", true);
        SelectOption opt2 = list.addOption("2", "two");
        SelectOption opt3 = list.addOption("3", "three");
        SelectOption opt4 = list.addOption("4", "four");
        SelectOption opt5 = new SelectOption("5+", "FiveOrMore", false);
        list.addOption(opt5);

        // Aggiungere elementi al layout di griglia
        grid3.addElement(new LineLayoutFormPanel());
        grid3.addElement(txt);
        grid3.addElement(list);
//-------------------------------------------------------------------

//-------------------------------------------------------------------
        // Creare un layout di griglia per commenti sul prodotto
        GridLayoutFormPanel grid4 = new GridLayoutFormPanel(1);
       txt = new HTMLText("Product Comments:");
        txt.setSize(4);

        // Aggiungere elementi al layout di griglia
        grid4.addElement(new LineLayoutFormPanel());
        grid4.addElement(txt);
        // Creare un modulo area testo
        grid4.addElement(new TextAreaFormElement("comments", 5, 75));
        grid4.addElement(new LineLayoutFormPanel());
//-------------------------------------------------------------------

//-------------------------------------------------------------------
        // Creare un layout di griglia
        GridLayoutFormPanel grid5 = new GridLayoutFormPanel(2);
txt = new HTMLText("Would you like to sign on to a server?");
        txt.setSize(4);

        // Creare un'immissione testo ed un'etichetta per il nome di sistema.
        TextFormInput sys = new TextFormInput("system");
        LabelFormElement sysLabel = new LabelFormElement("System:");

        // Creare un'immissione testo ed un'etichetta per l'id utente.
        TextFormInput uid = new TextFormInput("uid");
        LabelFormElement uidLabel = new LabelFormElement("UserID");

        // Creare un'immissione parola d'ordine ed un'etichetta per la parola d'ordine.
        PasswordFormInput pwd = new PasswordFormInput("pwd");
        LabelFormElement pwdLabel = new LabelFormElement("Password");

        // Aggiungere le immissioni testo, le immissioni parola d'ordine e le etichette alla griglia
```

```
                grid5.addElement(sysLabel);
                grid5.addElement(sys);
                grid5.addElement(uidLabel);
                grid5.addElement(uid);
                grid5.addElement(pwdLabel);
                grid5.addElement(pwd);
        //-----------------------------------------------------------------

        //-----------------------------------------------------------------
        // Aggiungere i vari pannelli creati al modulo HTML
        // nell'ordine in cui si desidera visualizzarli
                form.addElement(line2);
                 form.addElement(line3);
                 form.addElement(grid1);
                 form.addElement(grid2);
                 form.addElement(line4);
                 form.addElement(grid3);
                 form.addElement(grid4);
                 form.addElement(txt);
                 form.addElement(new LineLayoutFormPanel());
                 form.addElement(grid5);
                 form.addElement(new LineLayoutFormPanel());
        form.addElement(
           new HTMLText("Submit an attachment Here: <br />"));
                 // Aggiungere un'immissione file al modulo
                 form.addElement(new FileFormInput("myAttachment"));
        form.addElement(new ButtonFormInput("button",
                                            "TRY ME!",
                                            "test()"));
                 // Aggiunge un layout di riga, che a sua volta
        // aggiunge un'interruzione di riga <br /> al modulo
                 form.addElement(new LineLayoutFormPanel());
                 form.addElement(new LineLayoutFormPanel());
                 form.addElement(new SubmitFormInput("submit", "Register"));
                 form.addElement(new LineLayoutFormPanel());
                 form.addElement(new LineLayoutFormPanel());
        form.addElement(new ResetFormInput("reset", "Reset"));
                 // Aggiungere un'immissione nascosta al modulo
        form.addElement(new
          HiddenFormInput("copyright",
                          "(C) Copyright IBM Corp. 1999, 1999"));
        //-----------------------------------------------------------------

                 // Aggiungere l'intero Modulo HTML al buffer di stringa
               page.append(form.toString());

      }
        catch(Exception e)
      {
                 e.printStackTrace();
      }

            // Aggiungere le tag HTML finali al buffer
      page.append("</BODY>\n");
      page.append("</HTML>\n");

            // Restituire l'intera stringa della pagina HTML
          return page.toString();

   }
}
```

Emissione dell’esempio della classe HTML

Queste sono alcune emissioni di esempio che è possibile richiamare eseguendo l’Esempio della classe HTML:

- Customer Name: Fred Flinstone
  Email address: flinstone@bedrock.com
  Currently Using Toolbox: yes
  Requested More Information: yes
  Multiple Versions: v4r2,v4r4
  Using Java or Interested In: applications,servlets
  Platforms: NT,Linux
  Number of iSeries servers: three
  Comments: The Toolbox is being used by our entire Programming department to build customer applications!
  Attachment File: U:\wiedrich\servlet\temp.html

- Customer Name: Barney Rubble
  Email address: rubble@bedrock.com
  Currently Using Toolbox: yes
  AS400 Version: v4r4
  Using Java or Interested In: servlets
  Platforms: OS2
  Number of iSeries servers: FiveOrMore

- Customer Name: George Jetson
  Email address: jetson@sprocket.com
  Requested More Information: yes
  AS400 Version: v4r2
  Using Java or Interested In: applications
  Platforms: NT,Other
  Other Platforms: Solaris
  Number of iSeries servers: one
  Comments: This is my first time using this! Very Cool!

- Customer Name: Clark Kent
  Email address: superman@krypton.com
  AS400 Version: v4r2
  Number of iSeries servers: one

Esempio: utilizzo delle classi HTMLTree

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per informazioni legali importanti.

```
////////////////////////////////////////////////////////////////////////////
//
// Questo sorgente è un esempio di utilizzo delle classi pacchetto HTML di IBM Toolbox,
// che consentono di creare facilmente alberi HTML e File.
//
////////////////////////////////////////////////////////////////////////////

import java.io.File;
import java.io.PrintWriter;
import java.io.IOException;

import java.util.Vector;
import java.util.Properties;

import javax.servlet.*;
import javax.servlet.http.*;

import com.ibm.as400.access.AS400;
import com.ibm.as400.access.Trace;
```

```
import com.ibm.as400.access.IFSJavaFile;
import com.ibm.as400.util.html.HTMLMeta;
import com.ibm.as400.util.html.HTMLTree;
import com.ibm.as400.util.html.HTMLTreeElement;
import com.ibm.as400.util.html.URLParser;
import com.ibm.as400.util.html.DirFilter;
import com.ibm.as400.util.html.FileTreeElement;
import com.ibm.as400.util.servlet.ServletHyperlink;

/**
  *  Un esempio di utilizzo delle classi HTMLTree e FileTreeElement in un servlet.
 **/
public class TreeNav extends HttpServlet
{
      public void init(ServletConfig config)
            throws ServletException
   {
            super.init(config);

      // Il Toolbox utilizza una serie di icone predefinite per rappresentare elementi espansi,
      // compressi e documenti all'interno di HTMLTree. Per migliorare le icone,
      // il Toolbox invia tre file .gif (expanded.gif, collapsed.gif, bullet.gif)
      // al file jt400Servlet.jar. I browser non sono in grado di
      // rilevare i file .gif in un file .jar o .zip, pertanto è necessario estrarre tali immagini
      // dal file .jar e posizionarle nell'indirizzario del server web appropriato
      // (per impostazione predefinita è l'indirizzario /html). Quindi, eliminare il commento delle seguenti
      // righe di codice e specificare l'ubicazione corretta in questa serie di
      // metodi. L'ubicazione può essere assoluta o relativa.

      HTMLTreeElement.setExpandedGif("/images/expanded.gif");
      HTMLTreeElement.setCollapsedGif("/images/collapsed.gif");
      HTMLTreeElement.setDocGif("/images/bullet.gif");
   }

   /**
        *  Process the GET request.
        *  @param req The request.
        *  @param res The response.
    **/
      public void doGet (HttpServletRequest req, HttpServletResponse resp)
      throws ServletException, IOException
   {
            HttpSession session = req.getSession(true);
            HTMLTree fileTree = (HTMLTree)session.getValue("filetree");

            // Se questa sessione non dispone già di un albero file,
            // creare l'albero iniziale.
            if (fileTree == null)
                  fileTree = createTree(req, resp, req.getRequestURI());

            // Impostare la richiesta Http servlet nell'HTMLTree.
            fileTree.setHttpServletRequest(req);

            resp.setContentType("text/html");

            PrintWriter out = resp.getWriter();
      out.println("<html>\n");
            out.println(new HTMLMeta("Expires","Mon, 03 Jan 1990 13:00:00 GMT"));
      out.println("<body>\n");

            // Richiamare la tag per l'HTMLTree.
            out.println(fileTree.getTag());

      out.println("</body>\n");
      out.println("</html>\n");
            out.close();
```

```
   // Impostare il valore dell'albero di sessione, in modo tale che quando si immette questo servlet per
   // la seconda volta, l'oggetto FileTree verrà riutilizzato.
         session.putValue("filetree", fileTree);
}

/**
     *  Process the POST request.
     *  @param req The request.
     *  @param res The response.
 **/
   public void doPost (HttpServletRequest req, HttpServletResponse res)
   throws ServletException, IOException
{
         res.setContentType("text/html");
         ServletOutputStream out = res.getOutputStream();
}

/**
     *  Questo metodo creerà l'HTMLTree iniziale.
 **/
private HTMLTree createTree(HttpServletRequest req, HttpServletResponse resp, String uri)
{
   // Creare un oggetto HTMLTree.
         HTMLTree tree = new HTMLTree(req);

                              try
   {
      // Creare un oggetto URLParser.
                URLParser urlParser = new URLParser(uri);

      AS400 sys = new AS400(CPUStatus.systemName_, "javactl", "jteam1");

      // Creare un oggetto File ed impostare l'indirizzario IFS principale.
      IFSJavaFile root = new IFSJavaFile(sys, "/QIBM");

      // Creare un filtro ed elencare tutti gli indirizzari.
                DirFilter filter = new DirFilter();
      //File[] dirList = root.listFiles(filter);

      // Richiamare l'elenco di file che corrispondono al filtro dell'indirizzario.
                String[] list = root.list(filter);

                File[] dirList = new File[list.length];

      // Non si considera necessario che i server web utilizzino JDK1.2 poiché
      // la maggior parte delle JVM dei server web sono più lente nell'aggiornamento
      // all'ultimo livello di JDK.
      // Il modo più efficace di creare questi oggetti file è quello di utilizzare
      // il metodo listFiles(filter) in JDK1.2 che verrà eseguito
      // nel seguente modo, piuttosto che utilizzare il metodo list(filter)
      // e quindi convertire le schiere di stringhe restituite in appropriate
      // schiera di file.
      // File[] dirList = root.listFiles(filter);

                for (int j=0; j<dirList.length; ++j)
      {
                     if (root instanceof IFSJavaFile)
                           dirList[j] = new IFSJavaFile((IFSJavaFile)root, list[j]);
                                 else
                           dirList[j] = new File(list[j]);
      }

                for (int i=0; i<dirList.length; i++)
      {
         // Creare un FileTreeElement per ogni indirizzario nell'elenco.
                     FileTreeElement node = new FileTreeElement(dirList[i]);
```

```
            // Creare un ServletHyperlink per le icone di espansione/compressione.
                        ServletHyperlink sl = new ServletHyperlink(urlParser.getURI());
                        //sl.setHttpServletResponse(resp);
                        node.setIconUrl(sl);

            // Creare un ServletHyperlink nel servlet TreeList, che
            // visualizzerà il contenuto del FileTreeElement (indirizzario).
                        ServletHyperlink tl = new ServletHyperlink("/servlet/TreeList");
                        tl.setTarget("list");

            // Se il ServletHyperlink non è provvisto di nome, impostarlo sul
            // nome dell'indirizzario.
                        if (tl.getText() == null)
                             tl.setText(dirList[i].getName());

            // Impostare il TextUrl per il FileTreeElement.
                        node.setTextUrl(tl);

            // Aggiungere il FileTreeElement all'HTMLTree.
                        tree.addElement(node);
          }
        }
                                    catch (Exception e)
        {
          e.printStackTrace();
        }

                return tree;
    }

        public void destroy(ServletConfig config)
    {
            // nessuna operazione
    }

    public String getServletInfo()
    {
                return "FileTree Navigation";
    }
}
```

Esempio: creazione di una gerarchia ad albero IFS (file uno di tre)

Questo codice di esempio, unito al codice negli altri due file di esempio, visualizza un HTMLTree e un FileListElement in un servlet. I tre file nell'esempio sono:

- FileTreeExample.java - questo file genera le frame HTML e avvia il servlet
- TreeNav.java - crea e gestisce l'albero
- TreeList.java - visualizza il contenuto delle selezioni effettuate nella classe TreeNav.java

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per informazioni legali importanti.

```
////////////////////////////////////////////////////////////////////////////
//
// Questo sorgente è un esempio di utilizzo delle classi del pacchetto HTML di IBM Toolbox per Java,
// che consentono di creare facilmente alberi HTML e File.
//
////////////////////////////////////////////////////////////////////////////

import java.io.PrintWriter;
import java.io.IOException;

import javax.servlet.*;
import javax.servlet.http.*;

import com.ibm.as400.util.html.HTMLMeta;
```

```
//
// Un esempio di utilizzo delle frame per visualizzare un HTMLTree ed un FileListElement
// in un servlet.
//

public class FileTreeExample extends HttpServlet
{
      public void init(ServletConfig config)
            throws ServletException
   {
      super.init(config);
   }

   /**
        *  Process the GET request.
        *  @param req The request.
        *  @param res The response.
    **/

      public void doGet (HttpServletRequest req, HttpServletResponse resp)
      throws ServletException, IOException
   {
            resp.setContentType("text/html");

      // Impostare due frame. La prima, una frame di navigazione, visualizzerà
            // l'HTMLTree, il quale conterrà FileTreeElement e consentirà
      // la navigazione del File system. La seconda frame visualizzerà/elencherà
            // il contenuto di un indirizzario selezionato dalla frame di navigazione.
            PrintWriter out = resp.getWriter();
      out.println("<html>\n");
            out.println(new HTMLMeta("Expires","Mon, 04 Jan 1990 13:00:00 GMT"));
      out.println("<frameset cols=\"25%,*\">");
      out.println("<frame frameborder=\"5\" src=\"/servlet/TreeNav\" name=\"nav\">");
      out.println("<frame frameborder=\"3\" src=\"/servlet/TreeList\" name=\"list\">");
      out.println("</frameset>");
      out.println("</html>\n");
            out.close();
   }

   /**
        *  Process the POST request.
        *  @param req The request.
        *  @param res The response.
    **/

      public void doPost (HttpServletRequest req, HttpServletResponse res)
      throws ServletException, IOException
   {
            res.setContentType("text/html");
            ServletOutputStream out = res.getOutputStream();
   }

       public void destroy(ServletConfig config)
   {
            // nessuna operazione
   }

   public String getServletInfo()
   {
            return "FileTree Servlet";
   }
}
```

Esempio: creazione di una gerarchia ad albero IFS (File due di tre)

Questo codice di esempio, unito al codice negli altri due file di esempio, visualizza un HTMLTree e un FileListElement in un servlet. I tre file nell'esempio sono:

- FileTreeExample.java - crea le frame HTML e avvia il servlet
- TreeNav.java - questo file, che crea e gestisce l'albero
- TreeList.java - visualizza il contenuto delle selezioni effettuate nella classe TreeNav.java

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
////////////////////////////////////////////////////////////////////////////
//
// Questo sorgente è un esempio di utilizzo delle classi pacchetto HTML di IBM Toolbox,
// che consentono di creare facilmente alberi HTML e File.
//
////////////////////////////////////////////////////////////////////////////

import java.io.File;
import java.io.PrintWriter;
import java.io.IOException;

import javax.servlet.*;
import javax.servlet.http.*;

import com.ibm.as400.access.AS400;
import com.ibm.as400.access.IFSJavaFile;
import com.ibm.as400.util.html.HTMLMeta;
import com.ibm.as400.util.html.HTMLTree;
import com.ibm.as400.util.html.HTMLTreeElement;
import com.ibm.as400.util.html.URLParser;
import com.ibm.as400.util.html.DirFilter;
import com.ibm.as400.util.html.FileTreeElement;
import com.ibm.as400.util.servlet.ServletHyperlink;

//
// Un esempio di utilizzo delle classi HTMLTree e FileTreeElement
// in un servlet.
//

public class TreeNav extends HttpServlet
{
      private AS400 sys_;

      public void init(ServletConfig config)
            throws ServletException
   {
            super.init(config);

            // Creare un oggetto AS400.
            sys_ = new AS400("mySystem", "myUserID", "myPassword");

      // IBM Toolbox per Java utilizza una serie di icone predefinite per rappresentare elementi espansi,
      // ridotti e documenti nell'HTMLTree. Per migliorare queste icone,
      // IBM Toolbox per Java invia tre gif (expanded.gif, collapsed.gif, bullet.gif)
      // nel file jt400Servlet.jar.  I browser non hanno la capacità di individuare
      // gif in un file jar o zip, quindi è necessario estrarre tali immagini dal
      // file jar e collocarle nell'indirizzario server web appropriato (per impostazione predefinita
      // è l'indirizzario /html). Quindi modificare le seguenti righe di codice per
      // specificare l'ubicazione corretta nei metodi set.  L'ubicazione può essere
      // assoluta o relativa.

            HTMLTreeElement.setExpandedGif("http://myServer/expanded.gif");
            HTMLTreeElement.setCollapsedGif("http://myServer/collapsed.gif");
            HTMLTreeElement.setDocGif("http://myServer/bullet.gif");
   }
```

```
/**
     *  Process the GET request.
     *  @param req The request.
     *  @param res The response.
 **/

   public void doGet (HttpServletRequest req, HttpServletResponse resp)
         throws ServletException, IOException
{
         // Utilizzare i dati di sessione per ricordare lo stato dell'albero.
         HttpSession session = req.getSession(true);
         HTMLTree fileTree = (HTMLTree)session.getValue("filetree");

         // Se questa sessione non dispone già di un albero file,
         // creare l'albero iniziale.
         if (fileTree == null)
               fileTree = createTree(req, resp, req.getRequestURI());

         // Impostare la richiesta Http servlet nell'HTMLTree.
         fileTree.setHttpServletRequest(req);

         resp.setContentType("text/html");

         PrintWriter out = resp.getWriter();
   out.println("<html>\n");
         out.println(new HTMLMeta("Expires","Mon, 03 Jan 1990 13:00:00 GMT"));
   out.println("<body>\n");

         // Richiamare la tag per l'HTMLTree.
         out.println(fileTree.getTag());

   out.println("</body>\n");
   out.println("</html>\n");
         out.close();

   // Impostare il valore dell'albero di sessione, in modo tale che quando si immette questo servlet per
   // la seconda volta, l'oggetto FileTree verrà riutilizzato.
         session.putValue("filetree", fileTree);
}

/**
     *  Process the POST request.
     *  @param req The request.
     *  @param res The response.
 **/

   public void doPost (HttpServletRequest req, HttpServletResponse res)
         throws ServletException, IOException
{
         res.setContentType("text/html");
         ServletOutputStream out = res.getOutputStream();
}

/**
     *  Questo metodo creerà l'HTMLTree iniziale.
 **/

private HTMLTree createTree(HttpServletRequest req,
                            HttpServletResponse resp, String uri)
{
         // Creare un oggetto HTMLTree.
         HTMLTree tree = new HTMLTree(req);

                                 try
     {
                // Creare un oggetto URLParser.
```

```
                URLParser urlParser = new URLParser(uri);

               // Creare un oggetto File ed impostare l'indirizzario IFS principale.
                IFSJavaFile root = new IFSJavaFile(sys_, "/QIBM");

                // Creare un Filtro.
                DirFilter filter = new DirFilter();

               // Richiamare l'elenco di file che corrispondono al filtro dell'indirizzario.
                String[] list = root.list(filter);

                File[] dirList = new File[list.length];

        // Non si considera necessario che i server web utilizzino JDK1.2 poiché
        // la maggior parte delle JVM dei server Web sono più lente nell'aggiornamento all'ultimo
        // livello di JDK. Il modo più efficiente per creare questi oggetti file
        // è quello di utilizzare il metodo listFiles(filter) in JDK1.2 che dovrebbe
        // essere fatto nel modo seguente, invece di utilizzare il metodo list(filter)
        // e convertire quindi le schiere di stringhe restituite nella
        // schiera File appropriata.
        // File[] dirList = root.listFiles(filter);

                for (int j=0; j<dirList.length; ++j)
        {
                    if (root instanceof IFSJavaFile)
                          dirList[j] = new IFSJavaFile((IFSJavaFile)root, list[j]);
                                else
                          dirList[j] = new File(list[j]);
        }

                for (int i=0; i<dirList.length; i++)
        {
                    // Creare un FileTreeElement per ogni indirizzario nell'elenco.
                    FileTreeElement node = new FileTreeElement(dirList[i]);

                    // Creare un ServletHyperlink per le icone di espansione/compressione.
                    ServletHyperlink sl = new ServletHyperlink(urlParser.getURI());
                    sl.setHttpServletResponse(resp);
                    node.setIconUrl(sl);

                    // Creare un ServletHyperlink nel servlet TreeList, che
                    // visualizzerà il contenuto del FileTreeElement (indirizzario).
                    ServletHyperlink tl = new ServletHyperlink("/servlet/TreeList");
                    tl.setTarget("list");

                    // Se il ServletHyperlink non è provvisto di nome, impostarlo sul
                    // nome dell'indirizzario.
                    if (tl.getText() == null)
                        tl.setText(dirList[i].getName());

                    // Impostare il TextUrl per il FileTreeElement.
                    node.setTextUrl(tl);

                    // Aggiungere il FileTreeElement all'HTMLTree.
                    tree.addElement(node);
        }

                sys_.disconnectAllServices();
    }

                                catch (Exception e)
    {
             e.printStackTrace();
    }

            return tree;
}
```

```
        public void destroy(ServletConfig config)
    {
          // nessuna operazione
    }

        public String getServletInfo()
    {
            return "FileTree Navigation";
    }
}
```

Esempio: creazione di una gerarchia ad albero IFS (File tre di tre)

Questo codice di esempio, unito al codice negli altri due file di esempio, visualizza un HTMLTree e un FileListElement in un servlet. I tre file nell'esempio sono:

- FileTreeExample.java - crea le frame HTML e avvia il servlet
- TreeNav.java - crea e gestisce l'albero
- TreeList.java - questo file, che visualizza il contenuto delle selezioni effettuate nella classe TreeNav.java

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
///////////////////////////////////////////////////////////////////////////
//
// Questo sorgente è un esempio di utilizzo delle classi pacchetto HTML di IBM Toolbox,
// che consente di creare facilmente Elenchi HTML e File.
//
///////////////////////////////////////////////////////////////////////////

import java.io.PrintWriter;
import java.io.IOException;
import java.io.File;

import com.ibm.as400.access.AS400;
import com.ibm.as400.access.Trace;
import com.ibm.as400.access.IFSJavaFile;
import com.ibm.as400.util.html.HTMLMeta;
import com.ibm.as400.util.html.HTMLHeading;
import com.ibm.as400.util.html.HTMLConstants;
import com.ibm.as400.util.html.FileListElement;
import com.ibm.as400.util.html.*;

import javax.servlet.*;
import javax.servlet.http.*;

/**
  *  An example of using the FileListElement class in a servlet.
 **/
public class TreeList extends HttpServlet
{

       private AS400 sys_;

    /**
         *  Process the GET request.
         *  @param req The request.
         *  @param res The response.
     **/
       public void doGet (HttpServletRequest req, HttpServletResponse resp)
          throws ServletException, IOException
    {
              resp.setContentType("text/html");

                                  try
        {
```

```
            PrintWriter out = resp.getWriter();
        out.println("<html>\n");
        out.println(new HTMLMeta("Expires",
                                 "Mon, 02 Jan 1990 13:00:00 GMT"));
        out.println("<body>\n");

        // Se il parametro percorso non è nullo, l'utente ha selezionato un
        // elemento dall'elenco FileTreeElement nella frame di navigazione.
                    if (req.getPathInfo() != null)
        {
            // Creare un FileListElement passando un oggetto sistema AS400 e
            // una richiesta servlet Http. La richiesta conterrà le informazioni
            // sul percorso necessarie per elencare il contenuto del FileTreeElement
            // (indirizzario) selezionato.
                            FileListElement fileList = new FileListElement(sys_, req);

            //  In alternativa, creare un FileListElement da un nome di condivisione e da un percorso
            //  di condivisione NetServer.
            //  FileListElement fileList =
                  new FileListElement(sys_, req, "TreeShare",
                                      "/QIBM/ProdData/HTTP/Public/jt400");


                            // Visualizzare il contenuto del FileListElement.
                            out.println(fileList.list());
        }
        // Visualizzare tale HTMLHeading se non è stato selezionato alcun FileTreeElement.
                                else
        {
            HTMLHeading heading = new
              HTMLHeading(1,"An HTML File List Example");
                            heading.setAlign(HTMLConstants.CENTER);

                            out.println(heading.getTag());
        }

        out.println("</body>\n");
        out.println("</html>\n");
              out.close();
    }
                                catch (Exception e)
    {
                e.printStackTrace();
    }

}

/**
      *  Process the POST request.
      *  @param req The request.
      *  @param res The response.
**/
   public void doPost (HttpServletRequest req, HttpServletResponse res)
      throws ServletException, IOException
{
          res.setContentType("text/html");
          ServletOutputStream out = res.getOutputStream();

}

   public void init(ServletConfig config)
      throws ServletException
{
          super.init(config);
```

```
            // Creare un oggetto AS400.
        sys_ = new AS400("mySystem", "myUID", "myPWD");
    }
}
```

Esempio: utilizzo delle classi HTMLTable

L'esempio che segue mostra come funzionano le classi HTMLTable:

```
// Creare un oggetto HTMLTable predefinito.
         HTMLTable table = new HTMLTable();

// Impostare gli attributi di tabella.
table.setAlignment(HTMLTable.CENTER);
table.setBorderWidth(1);

// Creare un oggetto HTMLTableCaption predefinito ed impostare il testo del titolo.
      HTMLTableCaption caption = new HTMLTableCaption();
caption.setElement("Customer Account Balances - January 1, 2000");

// Impostare il titolo.
      table.setCaption(caption);

// Creare le intestazioni di tabella ed aggiungerle alla tabella.
HTMLTableHeader account_header = new HTMLTableHeader(new HTMLText("ACCOUNT"));
HTMLTableHeader name_header = new HTMLTableHeader(new HTMLText("NAME"));
HTMLTableHeader balance_header = new HTMLTableHeader(new HTMLText("BALANCE"));

table.addColumnHeader(account_header);
table.addColumnHeader(name_header);
table.addColumnHeader(balance_header);

// Aggiunge righe alla tabella. Ogni record del cliente rappresenta una riga nella tabella.
int numCols = 3;
for (int rowIndex=0; rowIndex< numCustomers; rowIndex++)
{
   HTMLTableRow row = new HTMLTableRow();
   row.setHorizontalAlignment(HTMLTableRow.CENTER);

      HTMLText account = new HTMLText(customers[rowIndex].getAccount());
      HTMLText name = new HTMLText(customers[rowIndex].getName());
      HTMLText balance = new HTMLText(customers[rowIndex].getBalance());

   row.addColumn(new HTMLTableCell(account));
   row.addColumn(new HTMLTableCell(name));
   row.addColumn(new HTMLTableCell(balance));

      // Aggiungere la riga alla tabella.
   table.addRow(row);
}
System.out.println(table.getTag());
```

L'esempio codice Java precedente crea il codice HTML che segue:

```
<table align="center" border="1">
<caption>Customer Account Balances - January 1, 2000</caption>
<tr>
<th>ACCOUNT</th>
<th>NAME</th>
<th>BALANCE</th>
</tr>
<tr align="center">
<td>0000001</td>
<td>Customer1</td>
<td>100.00</td>
</tr>
<tr align="center">
```

```
<td>0000002</td>
<td>Customer2</td>
<td>200.00</td>
</tr>
<tr align="center">
<td>0000003</td>
<td>Customer3</td>
<td>550.00</td>
</tr>
</table>
```

La tabella che segue mostra come viene visualizzato il codice HTML precedente in un browser web.

Tabella 2. Saldi conto cliente - 1 Gennaio 2000

CONTO	NOME	SALDO
0000001	Customer1	100.00
0000002	Customer2	200.00
0000003	Customer3	550.00

Esempi: PCML (Program Call Markup Language)

I seguenti esempi utilizzano PCML per richiamare le API di OS/400 ed ognuno si collega ad una pagina che visualizza il sorgente PCML seguito da un programma Java.

- Esempio semplice di richiamo dei dati: mostra il sorgente PCML ed il programma Java necessari per richiamare le informazioni su un profilo utente sul server. L'API che viene richiamata è l'API *Richiamo informazioni utente* (**QSYRUSRI**).
- Richiamo di un elenco di informazioni: mostra il sorgente PCML ed il programma Java necessari per richiamare un elenco di utenti autorizzati sul server. L'API richiamata è l'API *Apertura elenco utente autorizzati* (**QGYOLAUS**). Questo esempio illustra come accedere ad una schiera di strutture restituite da un programma del server.
- Richiamo di dati multidimensionali : mostra il sorgente PCML e il programma Java necessari per richiamare un elenco di esportazioni NFS (Network File System) da un server.L'API che viene richiamata è l'API *Richiamo delle esportazioni NFS* (**QZNFRTVE**).Questo esempio illustra come accedere a schiere di strutture in una schiera di strutture.

Nota: l'autorizzazione appropriata per ogni esempio varia ma può includere autorizzazioni specifiche all'oggetto e autorizzazioni speciali. Per eseguire questi esempi, è necessario collegarsi con un profilo utente che disponga di autorizzazioni per:

- Richiamare l'API OS/400 nell'esempio
- Accedere alle informazioni richieste

Il seguente esonero di responsabilità si applica a tutti gli esempi IBM Toolbox per Java:

Esonero di responsabilità per gli esempi di codice

IBM fornisce una licenza non esclusiva per utilizzare ciò come esempio da cui creare funzioni simili personalizzate, in base a richieste specifiche.

Tutti gli esempi di codice forniti dall'IBM hanno la sola funzione illustrativa. Questi esempi non sono stati interamente testati in tutte le condizioni. L'IBM, perciò, non intende implicita alcuna garanzia di affidabilità, manutenibilità o funzionalità di questi programmi.

Tutti i programmi qui contenuti sono forniti ″COSI' COME SONO″ senza garanzie di alcun tipo. Sono espressamente smentite tutte le garanzie implicite di non violazione, di commerciabilità e idoneità per scopi specifici.

Esempio: esempio semplice di richiamo dei dati

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per informazioni legali importanti.

Questo esempio semplice è composto di due parti:

- Sorgente PCML per richiamare QSYRUSRI
- Sorgente programma Java per richiamare QSYRUSRI

Sorgente PCML per richiamare QSYRUSRI

```
<pcml version="1.0">

<!-- Sorgente PCML per la chiamata all'API "Richiamo informazioni utente" (QSYRUSRI) -->

  <!-- Formato USRI0150 - Sono disponibili altri formati -->
  <struct name="usri0100">
    <data name="bytesReturned"              type="int"    length="4"  usage="output"/>
    <data name="bytesAvailable"             type="int"    length="4"  usage="output"/>
    <data name="userProfile"                type="char"   length="10" usage="output"/>
    <data name="previousSignonDate"         type="char"   length="7"  usage="output"/>
    <data name="previousSignonTime"         type="char"   length="6"  usage="output"/>
    <data                                   type="byte"   length="1"  usage="output"/>
    <data name="badSignonAttempts"          type="int"    length="4"  usage="output"/>
    <data name="status"                     type="char"   length="10" usage="output"/>
    <data name="passwordChangeDate"         type="byte"   length="8"  usage="output"/>
    <data name="noPassword"                 type="char"   length="1"  usage="output"/>
    <data                                   type="byte"   length="1"  usage="output"/>
    <data name="passwordExpirationInterval" type="int"    length="4"  usage="output"/>
    <data name="datePasswordExpires"        type="byte"   length="8"  usage="output"/>
    <data name="daysUntilPasswordExpires"   type="int"    length="4"  usage="output"/>
    <data name="setPasswordToExpire"        type="char"   length="1"  usage="output"/>
    <data name="displaySignonInfo"          type="char"   length="10" usage="output"/>
  </struct>

  <!-- Programma QSYRUSRI e il relativo elenco parametri per richiamare il formato USRI0100 -->
  <program name="qsyrusri" path="/QSYS.lib/QSYRUSRI.pgm">
    <data name="receiver"         type="struct"   struct="usri0100"   usage="output"/>
    <data name="receiverLength"   type="int"      length="4"          usage="input" />
    <data name="format"           type="char"     length="8"          usage="input"    init="USRI0100"/>
    <data name="profileName"      type="char"     length="10"         usage="input"    init="*CURRENT"/>
    <data name="errorCode"        type="int"      length="4"          usage="input"    init="0"/>
  </program>

</pcml>
```

Sorgente programma Java per richiamare QSYRUSRI

```
import com.ibm.as400.data.ProgramCallDocument;
import com.ibm.as400.data.PcmlException;
import com.ibm.as400.access.AS400;
import com.ibm.as400.access.AS400Message;

// Programma di esempio che mostra come richiamare le API "Richiamo informazioni utente" (QSYRUSRI)
public class qsyrusri {

        public qsyrusri() {
    }

      public static void main(String[] argv)
    {
            AS400 as400System;         // com.ibm.as400.access.AS400
```

```
            ProgramCallDocument pcml;  // com.ibm.as400.data.ProgramCallDocument
            boolean rc = false;        // Codice di ritorno da ProgramCallDocument.callProgram()
            String msgId, msgText;     // Messaggi restituiti dal server
            Object value;              // Valore di ritorno dal ProgramCallDocument.getValue()

            System.setErr(System.out);

              // Creare AS400 senza parametri, verrà effettuata una richiesta all'utente
            as400System = new AS400();

                                    try
        {
                  // Eliminare il commento da quanto segue per richiamare le informazioni di debug
                  //com.ibm.as400.data.PcmlMessageLog.setTraceEnabled(true);

            System.out.println("Beginning PCML Example..");
            System.out.println("    Constructing ProgramCallDocument for QSYRUSRI API...");

                  // Creare ProgramCallDocument
                  // Il primo parametro è il sistema a cui collegarsi
            // Il secondo parametro è un nome risorsa pcml. In questo esempio,
            // il file PCML serializzato "qsyrusri.pcml.ser" o
            // il file sorgente PCML "qsyrusri.pcml" devono trovarsi nel classpath.
            pcml = new ProgramCallDocument(as400System, "qsyrusri");

                                   // Impostare i parametri di immissione.
I valori predefiniti di diversi parametri sono
            // specificati nel sorgente PCML. Non è necessario impostarli utilizzando il codice Java.
            System.out.println("    Setting input parameters...");
            pcml.setValue("qsyrusri.receiverLength",
                          new Integer((pcml.getOutputsize("qsyrusri.receiver"))));

                  // Richiesta di chiamata dell'API
                  // All'utente verrà richiesto di collegarsi al sistema
            System.out.println("    Calling QSYRUSRI API requesting information for the sign-on user.");
            rc = pcml.callProgram("qsyrusri");

                  // Se il codice di ritorno è false, si riceveranno messaggi dal server
                  if(rc == false)
            {
                              // Reperire l'elenco di messaggi server
                AS400Message[] msgs = pcml.getMessageList("qsyrusri");

                              // Iterare i messaggi e scriverli nell'emissione standard
                              for (int m = 0; m < msgs.length; m++)
                {
                                      msgId = msgs[m].getID();
                                      msgText = msgs[m].getText();
                    System.out.println("    " + msgId + " - " + msgText);
                }
                System.out.println("** Call to QSYRUSRI failed. See messages above **");
                        System.exit(0);
            }
                        // Il codice di ritorno era true, chiamata a QSYRUSRI riuscita
                  // Scrivere alcune dei risultati nell'emissione standard
                                    else
            {
                value = pcml.getValue("qsyrusri.receiver.bytesReturned");
                System.out.println("        Bytes returned:      " + value);
                value = pcml.getValue("qsyrusri.receiver.bytesAvailable");
                System.out.println("        Bytes available:     " + value);
                value = pcml.getValue("qsyrusri.receiver.userProfile");
                System.out.println("        Profile name:        " + value);
                value = pcml.getValue("qsyrusri.receiver.previousSignonDate");
                System.out.println("        Previous signon date:" + value);
                value = pcml.getValue("qsyrusri.receiver.previousSignonTime");
                System.out.println("        Previous signon time:" + value);
```

```
            }
        }
                catch (PcmlException e)
        {
                  System.out.println(e.getLocalizedMessage());
                    e.printStackTrace();
            System.out.println("*** Call to QSYRUSRI failed. ***");
                    System.exit(0);
        }

                System.exit(0);
        } // End main()

}
```

Esempio: richiamo di un elenco di informazioni

Questo esempio è composto da due parti:

- Sorgente PCML per richiamare QGYOLAUS
- Sorgente del programma Java per richiamare QGYOLAUS

Sorgente PCML per richiamare QGYOLAUS

```
<pcml version="1.0">

<!-- Sorgente PCML per la chiamata all'API "Apertura elenco utenti autorizzati" (QGYOLAUS) -->

  <!-- Formato AUTU0150 - Sono disponibili altri formati -->
  <struct name="autu0150">
    <data name="name"         type="char" length="10" />
    <data name="userOrGroup"  type="char" length="1"  />
    <data name="groupMembers" type="char" length="1"  />
    <data name="description"  type="char" length="50" />
  </struct>

  <!-- Struttura informazioni sull'elenco (comune per le API di tipo "Apertura elenco") -->
  <struct name="listInfo">
    <data name="totalRcds"    type="int"  length="4" />
    <data name="rcdsReturned" type="int"  length="4" />
    <data name="rqsHandle"    type="byte" length="4" />
    <data name="rcdLength"    type="int"  length="4" />
    <data name="infoComplete" type="char" length="1" />
    <data name="dateCreated"  type="char" length="7" />
    <data name="timeCreated"  type="char" length="6" />
    <data name="listStatus"   type="char" length="1" />
    <data                     type="byte" length="1" />
    <data name="lengthOfInfo" type="int"  length="4" />
    <data name="firstRecord"  type="int"  length="4" />
    <data                     type="byte" length="40" />
  </struct>

  <!-- Programma QGYOLAUS e il relativo elenco parametri per richiamare il formato AUTU0150 -->
  <program name="qgyolaus" path="/QSYS.lib/QGY.lib/QGYOLAUS.pgm" parseorder="listInfo receiver">
    <data   name="receiver"       type="struct" struct="autu0150" usage="output"
              count="listInfo.rcdsReturned" outputsize="receiverLength" />
    <data   name="receiverLength" type="int"    length="4"        usage="input" init="16384" />
    <data   name="listInfo"       type="struct" struct="listInfo" usage="output" />
    <data   name="rcdsToReturn"   type="int"    length="4"        usage="input" init="264" />
    <data   name="format"         type="char"   length="10"       usage="input" init="AUTU0150" />
    <data   name="selection"      type="char"   length="10"       usage="input" init="*USER" />
    <data   name="member"         type="char"   length="10"       usage="input" init="*NONE" />
    <data   name="errorCode"      type="int"    length="4"        usage="input" init="0" />

  </program>

  <!-- Programma QGYGTLE ha restituito ulteriori "record" dall'elenco
```

```
      creato da QGYOLAUS. -->
  <program name="qgygtle" path="/QSYS.lib/QGY.lib/QGYGTLE.pgm" parseorder="listInfo receiver">
    <data   name="receiver"       type="struct" struct="autu0150" usage="output"
            count="listInfo.rcdsReturned" outputsize="receiverLength" />
    <data   name="receiverLength" type="int"    length="4"        usage="input" init="16384" />
    <data   name="requestHandle"  type="byte"   length="4"        usage="input" />
    <data   name="listInfo"       type="struct" struct="listInfo" usage="output" />
    <data   name="rcdsToReturn"   type="int"    length="4"        usage="input" init="264" />
    <data   name="startingRcd"    type="int"    length="4"        usage="input" />
    <data   name="errorCode"      type="int"    length="4"        usage="input" init="0" />
  </program>

  <!-- Il programma QGYCLST ha chiuso l'elenco, liberando le risorse sul server -->
  <program name="qgyclst" path="/QSYS.lib/QGY.lib/QGYCLST.pgm" >
    <data   name="requestHandle"  type="byte"   length="4"        usage="input" />
    <data   name="errorCode"      type="int"    length="4"        usage="input" init="0" />
  </program>
</pcml>
```

Sorgente del programma Java per richiamare QGYOLAUS

```
import com.ibm.as400.data.ProgramCallDocument;
import com.ibm.as400.data.PcmlException;
import com.ibm.as400.access.AS400;
import com.ibm.as400.access.AS400Message;

// Programma di esempio che mostra come richiamare l'API "Richiamo elenco utenti autorizzati" (QGYOLAUS)
public class qgyolaus
{

    public static void main(String[] argv)
  {
        AS400 as400System;         // com.ibm.as400.access.AS400
        ProgramCallDocument pcml;  // com.ibm.as400.data.ProgramCallDocument
        boolean rc = false;        // Codice di ritorno da ProgramCallDocument.callProgram()
        String msgId, msgText;     // Messaggi restituiti dal server
        Object value;              // Valore di ritorno dal ProgramCallDocument.getValue()

        int[] indices = new int[1]; // Indici per il valore della schiera di accesso
        int nbrRcds,                // Numero di record restituiti da QGYOLAUS e QGYGTLE
                nbrUsers;               // Numero totale di utenti richiamati
        String listStatus;          // Stato dell'elenco sul server
        byte[] requestHandle = new byte[4];

        System.setErr(System.out);

          // Creare AS400 senza parametri, verrà effettuata una richiesta all'utente
        as400System = new AS400();

                                try
    {
            // Eliminare il commento da quanto segue per richiamare le informazioni di debug
            //com.ibm.as400.data.PcmlMessageLog.setTraceEnabled(true);

      System.out.println("Beginning PCML Example..");
      System.out.println("    Constructing ProgramCallDocument for QGYOLAUS API...");

            // Creare ProgramCallDocument
            // Il primo parametro è il sistema a cui collegarsi
      // Il secondo parametro è un nome risorsa pcml. In questo esempio, il file sorgente
      // PCML serializzato "qgyolaus.pcml.ser" o
      // PCML "qgyolaus.pcml" deve trovarsi nel classpath.
      pcml = new ProgramCallDocument(as400System, "qgyolaus");

            // Tutti i parametri di immissione hanno valori predefiniti specificati nel sorgente PCML.
            // Non è necessario impostarli utilizzando il codice Java.
```

```
      // Richiesta di chiamata dell'API
      // All'utente verrà richiesto di collegarsi al sistema
System.out.println("    Calling QGYOLAUS API requesting information for the sign-on user.");
rc = pcml.callProgram("qgyolaus");

      // Se il codice di ritorno è false, si riceveranno messaggi dal server
      if(rc == false)
{
                // Reperire l'elenco di messaggi server
  AS400Message[] msgs = pcml.getMessageList("qgyolaus");

                // Iterare i messaggi e scriverli nell'emissione standard
                for (int m = 0; m < msgs.length; m++)
  {
                        msgId = msgs[m].getID();
                        msgText = msgs[m].getText();
      System.out.println("    " + msgId + " - " + msgText);
  }
  System.out.println("** Call to QGYOLAUS failed. See messages above **");
          System.exit(0);
}
      // Il codice di ritorno era true, chiamata a QGYOLAUS riuscita
      // Scrivere alcune dei risultati nell'emissione standard
                        else
{
          boolean doneProcessingList = false;
  String programName = "qgyolaus";
          nbrUsers = 0;
          while (!doneProcessingList)
  {
    nbrRcds = pcml.getIntValue(programName + ".listInfo.rcdsReturned");
    requestHandle = (byte[]) pcml.getValue(programName + ".listInfo.rqsHandle");

              // Iterare l'elenco di utenti
              for (indices[0] = 0; indices[0] < nbrRcds; indices[0]++)
    {
         value = pcml.getValue(programName + ".receiver.name", indices);
         System.out.println("User:  " + value);

         value = pcml.getValue(programName + ".receiver.description", indices);
         System.out.println("\t\t" + value);
    }

              nbrUsers += nbrRcds;

              // Controllare se sono stati richiamati tutti gli utenti.
    // In caso contrario, sarà necessario effettuare chiamate a "Richiamo voci elenco" (QGYGTLE)
              // per richiamare gli utenti rimanenti nell'elenco.
    listStatus = (String) pcml.getValue(programName + ".listInfo.listStatus");
    if ( listStatus.equals("2")   // List is marked as "Complete"
      || listStatus.equals("3") ) // Or list is marked "Error building"
    {
                 doneProcessingList = true;
    }
                            else
    {
      programName = "qgygtle";

                 // Impostare i valori di immissione per QGYGTLE
      pcml.setValue("qgygtle.requestHandle", requestHandle);
      pcml.setIntValue("qgygtle.startingRcd", nbrUsers + 1);

      // Richiamare "Richiamo voci elenco" (QGYGTLE) per richiamare ulteriori utenti dall'elenco
      rc = pcml.callProgram("qgygtle");

            // Se il codice di ritorno è false, si riceveranno messaggi dal server
            if(rc == false)
```

```
          {
                        // Reperire l'elenco di messaggi server
            AS400Message[] msgs = pcml.getMessageList("qgygtle");

                        // Iterare i messaggi e scriverli nell'emissione standard
                        for (int m = 0; m < msgs.length; m++)
            {
                               msgId = msgs[m].getID();
                               msgText = msgs[m].getText();
                System.out.println("    " + msgId + " - " + msgText);
            }
            System.out.println("** Call to QGYGTLE failed. See messages above **");
                    System.exit(0);
          }
                      // Il codice di ritorno era true, chiamata a QGYGTLE riuscita

        }
       }
       System.out.println("Number of users returned:  " + nbrUsers);

       // Richiamare l'API "Chiusura elenco" (QGYCLST)
       pcml.setValue("qgyclst.requestHandle", requestHandle);
       rc = pcml.callProgram("qgyclst");
     }
   }
       catch(PcmlException e)
   {
           System.out.println(e.getLocalizedMessage());
     e.printStackTrace();
     System.out.println("*** Call to QGYOLAUS failed. ***");
             System.exit(0);
   }

           System.exit(0);
  }
}
```

Esempio: richiamo di dati multidimensionali

Questo esempio è composto da due parti:

- Sorgente PCML per richiamare QZNFRTVE
- Sorgente programma Java per richiamare QZNFRTVE

Sorgente PCML per richiamare QZNFRTVE

```
<pcml version="1.0">

  <struct name="receiver">
    <data name="lengthOfEntry"            type="int"  length="4" />
    <data name="dispToObjectPathName"     type="int"  length="4" />
    <data name="lengthOfObjectPathName"   type="int"  length="4" />
    <data name="ccsidOfObjectPathName"    type="int"  length="4" />
    <data name="readOnlyFlag"             type="int"  length="4" />
    <data name="nosuidFlag"               type="int"  length="4" />
    <data name="dispToReadWriteHostNames" type="int"  length="4" />
    <data name="nbrOfReadWriteHostNames"  type="int"  length="4" />
    <data name="dispToRootHostNames"      type="int"  length="4" />
    <data name="nbrOfRootHostNames"       type="int"  length="4" />
    <data name="dispToAccessHostNames"    type="int"  length="4" />
    <data name="nbrOfAccessHostNames"     type="int"  length="4" />
    <data name="dispToHostOptions"        type="int"  length="4" />
    <data name="nbrOfHostOptions"         type="int"  length="4" />
    <data name="anonUserID"               type="int"  length="4" />
    <data name="anonUsrPrf"               type="char" length="10" />
    <data name="pathName"                 type="char" length="lengthOfObjectPathName"
          offset="dispToObjectPathName" offsetfrom="receiver" />
```

```
    <struct name="rwAccessList" count="nbrOfReadWriteHostNames"
            offset="dispToReadWriteHostNames" offsetfrom="receiver">
      <data name="lengthOfEntry"          type="int"  length="4" />
      <data name="lengthOfHostName"       type="int"  length="4" />
      <data name="hostName"               type="char" length="lengthOfHostName" />
      <data                               type="byte" length="0"
            offset="lengthOfEntry" />
    </struct>

    <struct name="rootAccessList" count="nbrOfRootHostNames"
            offset="dispToRootHostNames" offsetfrom="receiver">
      <data name="lengthOfEntry"          type="int"  length="4" />
      <data name="lengthOfHostName"       type="int"  length="4" />
      <data name="hostName"               type="char" length="lengthOfHostName" />
      <data                               type="byte" length="0"
            offset="lengthOfEntry" />
    </struct>

    <struct name="accessHostNames" count="nbrOfAccessHostNames"
            offset="dispToAccessHostNames" offsetfrom="receiver" >
      <data name="lengthOfEntry"          type="int"  length="4" />
      <data name="lengthOfHostName"       type="int"  length="4" />
      <data name="hostName"               type="char" length="lengthOfHostName" />
      <data                               type="byte" length="0"
            offset="lengthOfEntry" />
    </struct>

    <struct name="hostOptions" offset="dispToHostOptions" offsetfrom="receiver" count="nbrOfHostOptions">
      <data name="lengthOfEntry"          type="int"  length="4" />
      <data name="dataFileCodepage"       type="int"  length="4" />
      <data name="pathNameCodepage"       type="int"  length="4" />
      <data name="writeModeFlag"          type="int"  length="4" />
      <data name="lengthOfHostName"       type="int"  length="4" />
      <data name="hostName"               type="char" length="lengthOfHostName" />
      <data                               type="byte" length="0" offset="lengthOfEntry" />
    </struct>

    <data type="byte" length="0" offset="lengthOfEntry" />
  </struct>

  <struct name="returnedRcdsFdbkInfo">
    <data name="bytesReturned"            type="int" length="4" />
    <data name="bytesAvailable"           type="int" length="4" />
    <data name="nbrOfNFSExportEntries"    type="int" length="4" />
    <data name="handle"                   type="int" length="4" />
  </struct>

  <program name="qznfrtve" path="/QSYS.lib/QZNFRTVE.pgm" parseorder="returnedRcdsFdbkInfo receiver" >
    <data name="receiver"             type="struct" struct="receiver" usage="output"
             count="returnedRcdsFdbkInfo.nbrOfNFSExportEntries" outputsize="receiverLength"/>
    <data name="receiverLength"       type="int"    length="4" usage="input" init="4096" />
    <data name="returnedRcdsFdbkInfo" type="struct" struct="returnedRcdsFdbkInfo" usage="output" />
    <data name="formatName"           type="char"   length="8" usage="input" init="EXPE0100" />
    <data name="objectPathName"   type="char"   length="lengthObjPathName" usage="input" init="*FIRST" />
    <data name="lengthObjPathName"    type="int"    length="4" usage="input" init="6" />
    <data name="ccsidObjectPathName"  type="int"    length="4" usage="input" init="0" />
    <data name="desiredCCSID"         type="int"    length="4" usage="input" init="0" />
    <data name="handle"               type="int"    length="4" usage="input" init="0" />
    <data name="errorCode"            type="int"    length="4" usage="input" init="0" />
  </program>

</pcml>
```

Sorgente programma Java per richiamare QZNFRTVE

```
import com.ibm.as400.data.ProgramCallDocument;
import com.ibm.as400.data.PcmlException;
import com.ibm.as400.access.AS400;
import com.ibm.as400.access.AS400Message;

// Programma di esempio che mostra come richiamare l'API "Richiamo esportazioni NFS" (QZNFRTVE)
public class qznfrtve
{
    public static void main(String[] argv)
  {
        AS400 as400System;         // com.ibm.as400.access.AS400
        ProgramCallDocument pcml;  // com.ibm.as400.data.ProgramCallDocument
        boolean rc = false;        // Codice di ritorno da ProgramCallDocument.callProgram()
        String msgId, msgText;     // Messaggi restituiti dal server
        Object value;              // Valore di ritorno dal ProgramCallDocument.getValue()

        System.setErr(System.out);

          // Creare AS400 senza parametri, verrà effettuata una richiesta all'utente
        as400System = new AS400();

        int[] indices = new int[2]; // Indici per il valore della schiera di accesso
        int nbrExports;             // Numero di esportazioni restituite
    int nbrOfReadWriteHostNames, nbrOfRWHostNames, nbrOfRootHostNames,
        nbrOfAccessHostnames,    nbrOfHostOpts;

                                try
    {
            // Eliminare il commento da quanto segue per richiamare le informazioni di debug
            // com.ibm.as400.data.PcmlMessageLog.setTraceEnabled(true);

      System.out.println("Beginning PCML Example..");
      System.out.println("    Constructing ProgramCallDocument for QZNFRTVE API...");

            // Creare ProgramCallDocument
            // Il primo parametro è il sistema a cui collegarsi
      // Il secondo parametro è un nome risorsa pcml. In questo esempio, il file sorgente
      // PCML serializzato "qznfrtve.pcml.ser" o
      // PCML "qznfrtve.pcml" deve trovarsi nel classpath.
      pcml = new ProgramCallDocument(as400System, "qznfrtve");

                             // Impostare i parametri di immissione.
I valori predefiniti di diversi parametri sono
      // specificati nel sorgente PCML. Non è necessario impostarli utilizzando il codice Java.
      System.out.println("    Setting input parameters...");
      pcml.setValue("qznfrtve.receiverLength", new Integer( ( pcml.getOutputsize("qznfrtve.receiver"))));

            // Richiesta di chiamata dell'API
            // All'utente verrà richiesto di collegarsi al sistema
      System.out.println("    Calling QZNFRTVE API requesting NFS exports.");
      rc = pcml.callProgram("qznfrtve");

            if (rc == false)
      {
                    // Reperire l'elenco di messaggi server
        AS400Message[] msgs = pcml.getMessageList("qznfrtve");

                    // Iterare i messaggi e scriverli nell'emissione standard
                    for (int m = 0; m < msgs.length; m++)
        {
                              msgId = msgs[m].getID();
                              msgText = msgs[m].getText();
            System.out.println("    " + msgId + " - " + msgText);
        }
        System.out.println("** Call to QZNFRTVE failed. See messages above **");
                System.exit(0);
```

```
    }
          // Il codice di ritorno era true, chiamata a QZNFRTVE riuscita
          // Scrivere alcune dei risultati nell'emissione standard
                          else
    {
      nbrExports = pcml.getIntValue("qznfrtve.returnedRcdsFdbkInfo.nbrOfNFSExportEntries");
              // Iterare l'elenco delle esportazioni
              for (indices[0] = 0; indices[0] < nbrExports; indices[0]++)
      {
        value = pcml.getValue("qznfrtve.receiver.pathName", indices);
        System.out.println("Path name = " + value);

                  // Iterare e scrivere Read Write Host Names per questa esportazione
        nbrOfReadWriteHostNames = pcml.getIntValue("qznfrtve.receiver.nbrOfReadWriteHostNames",
                                                   indices);
                  for(indices[1] = 0; indices[1] < nbrOfReadWriteHostNames; indices[1]++)
        {
          value = pcml.getValue("qznfrtve.receiver.rwAccessList.hostName", indices);
          System.out.println("    Read/write access host name = " + value);
        }

                  // Iterare e scrivere Root Host Names per questa esportazione
        nbrOfRootHostNames = pcml.getIntValue("qznfrtve.receiver.nbrOfRootHostNames", indices);
                  for(indices[1] = 0; indices[1] < nbrOfRootHostNames; indices[1]++)
        {
          value = pcml.getValue("qznfrtve.receiver.rootAccessList.hostName", indices);
          System.out.println("    Root access host name = " + value);
        }

                  // Iterare e scrivere Access Host Names per questa esportazione
        nbrOfAccessHostnames = pcml.getIntValue("qznfrtve.receiver.nbrOfAccessHostNames",
                                                indices);
                  for(indices[1] = 0; indices[1] < nbrOfAccessHostnames; indices[1]++)
        {
          value = pcml.getValue("qznfrtve.receiver.accessHostNames.hostName", indices);
          System.out.println("    Access host name = " + value);
        }

                  // Iterare e scrivere Host Options per questa esportazione
        nbrOfHostOpts = pcml.getIntValue("qznfrtve.receiver.nbrOfHostOptions", indices);
                  for(indices[1] = 0; indices[1] < nbrOfHostOpts; indices[1]++)
        {
          System.out.println("    Host options:");
          value = pcml.getValue("qznfrtve.receiver.hostOptions.dataFileCodepage", indices);
          System.out.println("        Data file code page = " + value);
          value = pcml.getValue("qznfrtve.receiver.hostOptions.pathNameCodepage", indices);
          System.out.println("        Path name code page = " + value);
          value = pcml.getValue("qznfrtve.receiver.hostOptions.writeModeFlag", indices);
          System.out.println("        Write mode flag = " + value);
          value = pcml.getValue("qznfrtve.receiver.hostOptions.hostName", indices);
          System.out.println("        Host name = " + value);
        }
              } // termine elenco esportazioni di iterazioni loop
        } // termine chiamata a QZNFRTVE riuscita
    }
        catch(PcmlException e)
    {
            System.out.println(e.getLocalizedMessage());
              e.printStackTrace();
        System.exit(-1);
    }

            System.exit(0);
    } // end main()
}
```

Esempi: classi ReportWriter

Questa sezione elenca gli esempi di codice forniti in tutta la documentazione delle classi ReportWriter di IBM Toolbox per Java.

JSPReportProcessor e PDFContext

- Esempio: utilizzo di JSPReportProcessor con PDFContext
- Esempio: JSPReportProcessor file JSP di esempio

XSLReportProcessor e PCLContext

- Esempio: utilizzo di XSLReportProcessor con PCLContext
- Esempio: file XML di esempio XSLReportProcessor
- Esempio: file XML di esempio XSLReportProcessor

Il seguente esonero di responsabilità si applica a tutti gli esempi IBM Toolbox per Java:

> **Esonero di responsabilità per gli esempi di codice**
>
> L'IBM fornisce una licenza non esclusiva per utilizzare tutti gli esempi del codice di programmazione da cui creare funzioni simili personalizzate, in base a richieste specifiche.
>
> Questo codice di esempio è fornito dall'IBM con la sola funzione illustrativa. Questi esempi non sono stati interamente testati in tutte le condizioni. IBM, perciò, non fornisce nessun tipo di garanzia o affidabilità implicita, rispetto alla funzionalità o alle funzioni di questi programmi.
>
> Tutti i programmi qui contenuti vengono forniti all'utente "COSI' COME SONO" senza garanzie di alcun tipo. Le garanzie implicite di non contraffazione, commerciabilità e adeguatezza a scopi specifici sono espressamente vietate.

Esempio: utilizzo di JSPReportProcessor con PDFContext

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per informazioni legali importanti.

Per visualizzare il contenuto di un file sorgente di esempio JSP che è possibile utilizzare con JSPRunReport, consultare JSPcust_table.jsp. E' inoltre possibile scaricare un file ZIP che contiene il file di esempio JSP. Il file .zip contiene inoltre i file di esempio XML e XSL che è possibile utilizzare con l'esempio XSLReportProcessor (PCLRunReport).

```
////////////////////////////////////////////////////////////////////////////
//
// Il seguente esempio (JSPRunReport) utilizza le classi JSPReportProcessor e
// PDFContext per ottenere dati da un URL specificato e convertire i dati
// nel formato PDF. I dati vengono successivamente inviati a un file come documento PDF.
//
// Sintassi del comando:
//     java JSPRunReport <jsp_Url> <output_filename>
//
////////////////////////////////////////////////////////////////////////////

import java.lang.*;
import java.awt.*;
import java.io.*;
import java.net.*;
import java.awt.print.*;
import java.awt.event.*;
import java.util.LinkedList;
import java.util.ListIterator;
import java.util.HashMap;
```

```
import com.ibm.xsl.composer.flo.*;
import com.ibm.xsl.composer.areas.*;
import com.ibm.xsl.composer.framework.*;
import com.ibm.xsl.composer.java2d.*;
import com.ibm.xsl.composer.prim.*;
import com.ibm.xsl.composer.properties.*;
import com.ibm.as400.util.reportwriter.processor.*;
import com.ibm.as400.util.reportwriter.pdfwriter.*;
import java.io.IOException;
import java.io.Serializable;
import org.xml.sax.SAXException;


public class JSPRunReport

{

        public static void main(String args[])
    {
                FileOutputStream fileout = null;

        /** specificare la URL che contiene i dati che si desidera utilizzare nel prospetto **/
                String JSPurl = args[0];
                URL jspurl = null;
                try {
                            jspurl = new URL(JSPurl);
        }
                catch (MalformedURLException e)
        {}

                /** richiamare il nome del file PDF di emissione  **/
                String filename = args[1];
                try {
                fileout = new FileOutputStream(filename);
        }
                catch (FileNotFoundException e)
        {}

                /** impostare formato pagina **/
                Paper paper = new Paper();
                paper.setSize(612,792);
                paper.setImageableArea(18, 18, 576, 756);
                PageFormat pf = new PageFormat();
                pf.setPaper(paper);

        /** creare un oggetto PDFContext ed emettere FileOutputStream come OutputStream **/
                PDFContext pdfcontext = new PDFContext((OutputStream)fileout, pf);

                System.out.println( Ready to parse XSL document );

        /** creare l'oggetto JSPReportProcessor e impostare la mascherina al JSP specificato **/
                JSPReportProcessor jspprocessor = new JSPReportProcessor(pdfcontext);
                try {
                jspprocessor.setTemplate(jspurl);
        }

                catch (NullPointerException np){
                        String mes = np.getMessage();
                        System.out.println(mes);
                    System.exit(0);
            }

                /** elaborare il prospetto **/
                try {
                jspprocessor.processReport();
        }
                    catch (IOException e) {
```

```
                        String mes = e.getMessage();
                        System.out.println(mes);
                    System.exit(0);
            }
                catch (SAXException se) {
                        String mes = se.getMessage();
                        System.out.println(mes);
                    System.exit(0);
            }

                System.exit(0);
    }
}
```

Esempio: file JSP di esempio JSPReportProcessor

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per informazioni legali importanti.

```
<?xml version="1.0"?>

<!--
  Copyright (c) 1999 The Apache Software Foundation.  All rights reserved.
-->

<%@ page session="false"%>
<%@ page language="java" contentType="text/html" %>
<%@ page import="java.lang.*" %>
<%@ page import="java.util.*" %>


<%-- <jsp:useBean id='cust_table' scope='page' class='table.JSPcust_table'/> --%>

<%!
      String[][] cust_data = new String [4][5];

      public void jspInit()
   {
          //cust_record_field [][] cust_data;
        // cust_record holds customer name, customer address, customer city, customer state,
     // customer zip


     String [] cust_record_1 = {"IBM","3602 4th St","Rochester","Mn","55901"};
     String [] cust_record_2 = {"HP","400 2nd","Springfield","Mo","33559"};
     String [] cust_record_3 = {"Wolzack","34 Hwy 52N","Lansing","Or","67895"};
     String [] cust_record_4 = {"Siems","343 60th","Salem","Tx","12345"};

        cust_data[0] = cust_record_1;
        cust_data[1] = cust_record_2;
        cust_data[2] = cust_record_3;
        cust_data[3] = cust_record_4;
   }
%>

<!-- Prima verifica dell'analisi e della composizione. -->
<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
  <fo:layout-master-set>
    <fo:simple-page-master master-name="theMaster" >
    <fo:region-body region-name="theRegion" margin-left=".2in"/>
    </fo:simple-page-master>
    <fo:page-sequence-master master-name="theMaster">
    <fo:single-page-master-reference master-name="thePage"/>
    </fo:page-sequence-master>
  </fo:layout-master-set>
  <fo:page-sequence master-name="theMaster">
   <fo:flow flow-name="theRegion">
     <fo:block>
```

```
   <fo:block text-align="center"> NORCAP </fo:block>
   <fo:block space-before=".2in" text-align="center">PAN PACIFIC HOTEL IN SAN FRANCISCO </fo:block>
   <fo:block text-align="center"> FRIDAY, DECEMBER 8-9, 2000 </fo:block>
</fo:block>
<fo:block space-before=".5in" font-size="8pt">
<fo:table table-layout="fixed">
  <fo:table-column column-width="3in"/>
  <fo:table-column column-width="3in"/>
  <fo:table-column column-width="3in"/>
  <fo:table-column column-width="3in"/>
  <fo:table-column column-width="3in"/>
  <fo:table-body>
    <fo:table-row>
       <fo:table-cell column-number="1">
          <fo:block border-bottom-style="solid">NAME</fo:block>
       </fo:table-cell>
       <fo:table-cell column-number="2">
          <fo:block border-bottom-style="solid">ADDRESS</fo:block>
       </fo:table-cell>
       <fo:table-cell column-number="3">
          <fo:block border-bottom-style="solid">CITY</fo:block>
       </fo:table-cell>
       <fo:table-cell column-number="4">
          <fo:block border-bottom-style="solid">STATE</fo:block>
       </fo:table-cell>
       <fo:table-cell column-number="5">
          <fo:block border-bottom-style="solid">ZIP CODE</fo:block>
       </fo:table-cell>
    </fo:table-row>

    <%
                    // aggiungere una riga alla tabella
                    for(int i = 0; i <= 3; i++)
      {
                        String[] _array = cust_data[i];
    %>

    <fo:table-row>
      <fo:table-cell column-number="1">
        <fo:block space-before=".1in">
        <% if(_array[0].equals("IBM")) { %>
           <fo:inline background-color="blue">
           <% out.print(_array[0]); %>
           </fo:inline>
        <% } else { %>
           <% out.print(_array[0]); %>
        <% } %>
        </fo:block>
      </fo:table-cell>
      <fo:table-cell column-number="2">
        <fo:block space-before=".1in">
            <% out.print(_array[1]); %>
        </fo:block>
      </fo:table-cell>
      <fo:table-cell column-number="3">
        <fo:block space-before=".1in">
            <% out.print(_array[2]); %>
        </fo:block>
      </fo:table-cell>
      <fo:table-cell column-number="4">
        <fo:block space-before=".1in">
           <% out.print(_array[3]); %>
        </fo:block>
      </fo:table-cell>
      <fo:table-cell column-number="5">
       <fo:block space-before=".1in">
          <% out.print(_array[4]); %>
```

```
            </fo:block>
          </fo:table-cell>
         </fo:table-row>

       <%
                } // end row while
       %>

       </fo:table-body>
     </fo:table>
     </fo:block>
   </fo:flow>
  </fo:page-sequence>
</fo:root>
```

Esempio: utilizzo di XSLReportProcessor con PCLContext

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per informazioni legali importanti.

```
////////////////////////////////////////////////////////////////////////////
//
// Il seguente esempio (PCLRunReport) utilizza le classi XSLPReportProcessor e
// PCLContext per ottenere dati XML e convertire i dati in formato PCL.
// Il flusso di dati viene quindi inviato ad un OutputQueue di stampante.
//
// Per visualizzare il contenuto dei file sorgente XML e XSL di esempio che è possibile utilizzare
// con PCLRunReport, consultare realestate.xml e realestate.xsl. E' inoltre possibile
// scaricare un file .zip che i file di esempio XML e XSL. Il file
// .zip contiene inoltre un file di esempio JSP che è possibile utilizzare con
// l'esempio JSPReportProcessor (JSPRunReport).
//
// Sintassi del comando:
//     java PCLRunReport <xml_file> <xsl_file>
//
////////////////////////////////////////////////////////////////////////////

import java.lang.*;
import java.awt.*;
import java.io.*;
import java.awt.print.*;
import java.awt.event.*;
import java.util.LinkedList;
import java.util.ListIterator;
import java.util.HashMap;

import com.ibm.xsl.composer.flo.*;
import com.ibm.xsl.composer.areas.*;
import com.ibm.xsl.composer.framework.*;
import com.ibm.xsl.composer.java2d.*;
import com.ibm.xsl.composer.prim.*;
import com.ibm.xsl.composer.properties.*;
import com.ibm.as400.util.reportwriter.processor.*;
import com.ibm.as400.util.reportwriter.pclwriter.*;
import java.io.IOException;
import java.io.Serializable;
import org.xml.sax.SAXException;
import com.ibm.as400.access.*;

public class PCLRunReport

{

        public static void main(String args[])
    {
                SpooledFileOutputStream fileout = null;
                String xmldocumentName = args[0];
                String xsldocumentName = args[1];
```

```
String sys = "<system>";      /* Insert ISeries server name       */
String user = "<user>";       /* Insert ISeries user profile name */
String pass = "<password>";   /* Insert ISeries password          */

        AS400 system = new AS400(sys, user, pass);

        /* Insert ISeries output queue */
String outqname = "/QSYS.LIB/qusrsys.LIB/<outq>.OUTQ";
        OutputQueue outq = new OutputQueue(system, outqname);
            PrintParameterList parms = new PrintParameterList();
        parms.setParameter(PrintObject.ATTR_OUTPUT_QUEUE, outq.getPath());

        try{
            fileout = new SpooledFileOutputStream(system, parms, null, null);
}
                            catch (Exception e)
{}

        /** impostare formato pagina **/
        Paper paper = new Paper();
        paper.setSize(612,792);
        paper.setImageableArea(18, 36, 576, 720);
        PageFormat pf = new PageFormat();
        pf.setPaper(paper);

        /** creare un oggetto PCLContext ed incasellare FileOutputStream
                come un OutputStream **/
        PCLContext pclcontext = new PCLContext((OutputStream)fileout, pf);

System.out.println("Ready to parse XSL document");

        /** creare l'oggetto XSLReportProcessor **/
        XSLReportProcessor xslprocessor = new XSLReportProcessor(pclcontext);
        try {
        xslprocessor.setXMLDataSource(xmldocumentName);
}
        catch (SAXException se) {
                String mes = se.getMessage();
                System.out.println(mes);
    System.exit(0);
    }
        catch (IOException ioe) {
                String mes = ioe.getMessage();
                System.out.println(mes);
    System.exit(0);
    }
        catch (NullPointerException np){
                String mes = np.getMessage();
                System.out.println(mes);
    System.exit(0);
    }
        /** impostare la mascherina sull'origine dati XML specificata **/
        try {
        xslprocessor.setTemplate(xsldocumentName);
}
        catch (NullPointerException np){
                String mes = np.getMessage();
                System.out.println(mes);
    System.exit(0);
    }
            catch (IOException e) {
                String mes = e.getMessage();
                System.out.println(mes);
    System.exit(0);
    }
        catch (SAXException se) {
```

```
                        String mes = se.getMessage();
                        System.out.println(mes);
            System.exit(0);
            }

                /** elaborare il prospetto **/
                try {
                xslprocessor.processReport();
        }
                    catch (IOException e) {
                        String mes = e.getMessage();
                        System.out.println(mes);
            System.exit(0);
            }
                catch (SAXException se) {
                        String mes = se.getMessage();
                        System.out.println(mes);
            System.exit(0);
            }

        System.exit(0);
    }

}
```

Esempio: file XML di esempio XSLReportProcessor

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
<?xml version="1.0"?>

<RESIDENTIAL-LISTINGS VERSION="061698">

<RESIDENTIAL-LISTING ID="ID1287" VERSION="061698">
  <GENERAL>
 <TYPE>Apartment</TYPE>
 <PRICE>$110,000</PRICE>
 <STRUCTURE><NUM-BEDS>3</NUM-BEDS><NUM-BATHS>1</NUM-BATHS></STRUCTURE>
  <AGE UNITS="YEARS">15</AGE>
  <LOCATION COUNTRY="USA" STATE="MA" COUNTY="MIDDLESEX" SECURITY="Public">
   <ADDRESS>13 Some Avenue</ADDRESS>
    <CITY>Dorchester</CITY><ZIP>02121</ZIP>
 </LOCATION>
  <IMAGE FORMAT="JPG" WIDTH="300" HEIGHT="150" SRC="house1.jpg"/>
 <MLS>
    <MLS-CODE SECURITY="Restricted">
   30224877
  </MLS-CODE>
    <MLS-SOURCE SECURITY="Public">
   <NAME>Bob the Realtor</NAME>
   <PHONE>1-617-555-1212</PHONE>
   <FAX>1-617-555-1313</FAX>
   <WEB>
    <EMAIL>Bob@bigbucks.com</EMAIL>
    <SITE>www.bigbucks.com</SITE>
   </WEB>
  </MLS-SOURCE>
 </MLS>
 <DATES><LISTING-DATE>3/5/98</LISTING-DATE></DATES>
 <LAND-AREA UNITS="ACRES">0.01</LAND-AREA>

  </GENERAL>

  <FEATURES>
     <DISCLOSURES>
  In your dreams.
     </DISCLOSURES>
```

```
 <UTILITIES>
    Yes
 </UTILITIES>
 <EXTRAS>
    Pest control included.
 </EXTRAS>
 <CONSTRUCTION>
    Wallboard and glue
 </CONSTRUCTION>
 <ACCESS>
    Front door.
 </ACCESS>
 </FEATURES>

 <FINANCIAL>
 <ASSUMABLE>
    I assume so.
 </ASSUMABLE>
 <OWNER-CARRY>
    Too heavy.
 </OWNER-CARRY>
 <ASSESMENTS>
    $150,000
 </ASSESMENTS>
 <DUES>
    $100
 </DUES>
 <TAXES>
    $2,000
 </TAXES>
 <LENDER>
    Fly by nite mortgage co.
 </LENDER>
 <EARNEST>
    Burt
 </EARNEST>
 <DIRECTIONS>
    North, south, east, west
 </DIRECTIONS>
 </FINANCIAL>

 <REMARKS>
 </REMARKS>

 <CONTACTS>
<COMPANY>
 <NAME>
    Noplace Realty
 </NAME>
 <ADDRESS>
    12 Main Street
 </ADDRESS>
 <CITY>
    Lowell, MA
 </CITY>
 <ZIP>
    34567
 </ZIP>
</COMPANY>
<AGENT>
 <NAME>
    Mary Jones
 </NAME>
 <ADDRESS>
 </ADDRESS>
 <CITY>
 </CITY>
```

```
  <ZIP>
  </ZIP>
 </AGENT>
 <OWNER>
  <NAME>
  </NAME>
  <ADDRESS>
  </ADDRESS>
  <CITY>
  </CITY>
  <ZIP>
  </ZIP>
 </OWNER>
    <TENANT>
    Yes.
 </TENANT>
    <COMMISION>
    15%
 </COMMISION>
  </CONTACTS>

</RESIDENTIAL-LISTING>

<RESIDENTIAL-LISTING VERSION="061698" ID="ID1289">
  <GENERAL>

    <IMAGE FORMAT="JPG" WIDTH="300" HEIGHT="150" SRC="house2.jpg">
 </IMAGE>

 <MLS>
               <MLS-CODE SECURITY="Restricted">
      30298877
  </MLS-CODE>
               <MLS-SOURCE SECURITY="Public">
   <NAME>
    Mary the Realtor
   </NAME>
   <PHONE>
    1-617-555-3333
   </PHONE>
   <FAX>
    1-617-555-4444
   </FAX>
   <WEB>
    <EMAIL>
      Mary@somebucks.com
    </EMAIL>
    <SITE>
      www.bigbucks.com
    </SITE>
   </WEB>
  </MLS-SOURCE>
 </MLS>

 <TYPE>
  Home
 </TYPE>

 <PRICE>
  $200,000
 </PRICE>

        <AGE UNITS="MONTHS">
  3
 </AGE>

        <LOCATION COUNTRY="USA" STATE="CO" COUNTY="MIDDLESEX" SECURITY="Public">
```

```
 <ADDRESS>
  1 Main Street
 </ADDRESS>
 <CITY>
     Boulder
 </CITY>
 <ZIP>
  11111
 </ZIP>
</LOCATION>

<STRUCTURE>
  <NUM-BEDS>
    2
          </NUM-BEDS>
  <NUM-BATHS>
    2
          </NUM-BATHS>
</STRUCTURE>

<DATES>
 <LISTING-DATE>
     4/3/98
                </LISTING-DATE>
</DATES>

       <LAND-AREA UNITS="ACRES">
 0.01
</LAND-AREA>

 </GENERAL>

 <FEATURES>
    <DISCLOSURES>
 In your dreams.
    </DISCLOSURES>
 <UTILITIES>
    Yes
 </UTILITIES>
 <EXTRAS>
    Pest control included.
 </EXTRAS>
 <CONSTRUCTION>
    Wallboard and glue
 </CONSTRUCTION>
 <ACCESS>
    Front door.
 </ACCESS>
 </FEATURES>

 <FINANCIAL>
 <ASSUMABLE>
    I assume so.
 </ASSUMABLE>
 <OWNER-CARRY>
    Too heavy.
 </OWNER-CARRY>
 <ASSESMENTS>
    $150,000
 </ASSESMENTS>
 <DUES>
    $100
 </DUES>
 <TAXES>
    $2,000
 </TAXES>
 <LENDER>
```

```
     Fly by nite mortgage co.
  </LENDER>
  <EARNEST>
     Burt
  </EARNEST>
  <DIRECTIONS>
     North, south, east, west
  </DIRECTIONS>
  </FINANCIAL>

  <REMARKS>
  </REMARKS>

  <CONTACTS>
 <COMPANY>
  <NAME>
     Noplace Realty
  </NAME>
  <ADDRESS>
     12 Main Street
  </ADDRESS>
  <CITY>
     Lowell, MA
  </CITY>
  <ZIP>
     34567
  </ZIP>
 </COMPANY>
 <AGENT>
  <NAME>
     Mary Jones
  </NAME>
  <ADDRESS>
  </ADDRESS>
  <CITY>
  </CITY>
  <ZIP>
  </ZIP>
 </AGENT>
 <OWNER>
  <NAME>
  </NAME>
  <ADDRESS>
  </ADDRESS>
  <CITY>
  </CITY>
  <ZIP>
  </ZIP>
 </OWNER>
    <TENANT>
    Yes.
 </TENANT>
    <COMMISION>
    15%
 </COMMISION>
  </CONTACTS>

</RESIDENTIAL-LISTING>
<RESIDENTIAL-LISTING VERSION="061698" ID="ID1290">
  <GENERAL>

    <IMAGE FORMAT="JPG" WIDTH="300" HEIGHT="150" SRC="house3.jpg">
 </IMAGE>

 <MLS>
                <MLS-CODE SECURITY="Restricted">
   20079877
```

```
 </MLS-CODE>
              <MLS-SOURCE SECURITY="Public">
  <NAME>
   Bob the Realtor
  </NAME>
  <PHONE>
   1-617-555-1212
  </PHONE>
  <FAX>
   1-617-555-1313
  </FAX>
  <WEB>
   <EMAIL>
     Bob@bigbucks.com
   </EMAIL>
   <SITE>
     www.bigbucks.com
   </SITE>
  </WEB>
 </MLS-SOURCE>
</MLS>

<TYPE>
 Apartment
</TYPE>

<PRICE>
 $65,000
</PRICE>

       <AGE UNITS="YEARS">
 30
</AGE>

       <LOCATION COUNTRY="USA" STATE="MA" COUNTY="MIDDLESEX" SECURITY="Public">
 <ADDRESS>
  25 Which Ave.
 </ADDRESS>
 <CITY>
     Cambridge
 </CITY>
 <ZIP>
  02139
 </ZIP>
</LOCATION>

<STRUCTURE>
  <NUM-BEDS>
    3
         </NUM-BEDS>
  <NUM-BATHS>
    1
         </NUM-BATHS>
</STRUCTURE>

<DATES>
 <LISTING-DATE>
     3/5/97
               </LISTING-DATE>
</DATES>

       <LAND-AREA UNITS="ACRES">
 0.05
</LAND-AREA>

 </GENERAL>
```

```
 <FEATURES>
    <DISCLOSURES>
 In your dreams.
    </DISCLOSURES>
 <UTILITIES>
    Yes
 </UTILITIES>
 <EXTRAS>
    Pest control included.
 </EXTRAS>
 <CONSTRUCTION>
    Wallboard and glue
 </CONSTRUCTION>
 <ACCESS>
    Front door.
 </ACCESS>
 </FEATURES>

 <FINANCIAL>
 <ASSUMABLE>
    I assume so.
 </ASSUMABLE>
 <OWNER-CARRY>
    Too heavy.
 </OWNER-CARRY>
 <ASSESMENTS>
    $150,000
 </ASSESMENTS>
 <DUES>
    $100
 </DUES>
 <TAXES>
    $2,000
 </TAXES>
 <LENDER>
    Fly by nite mortgage co.
 </LENDER>
 <EARNEST>
    Burt
 </EARNEST>
 <DIRECTIONS>
    North, south, east, west
 </DIRECTIONS>
 </FINANCIAL>

 <REMARKS>
 </REMARKS>

 <CONTACTS>
<COMPANY>
 <NAME>
    Noplace Realty
 </NAME>
 <ADDRESS>
    12 Main Street
 </ADDRESS>
 <CITY>
    Lowell, MA
 </CITY>
 <ZIP>
    34567
 </ZIP>
</COMPANY>
<AGENT>
 <NAME>
    Mary Jones
 </NAME>
```

```
  <ADDRESS>
  </ADDRESS>
  <CITY>
  </CITY>
  <ZIP>
  </ZIP>
 </AGENT>
 <OWNER>
  <NAME>
  </NAME>
  <ADDRESS>
  </ADDRESS>
  <CITY>
  </CITY>
  <ZIP>
  </ZIP>
 </OWNER>
    <TENANT>
    Yes.
 </TENANT>
    <COMMISION>
    15%
 </COMMISION>
  </CONTACTS>

</RESIDENTIAL-LISTING>
<RESIDENTIAL-LISTING VERSION="061698"  ID="ID1291">
  <GENERAL>

    <IMAGE FORMAT="JPG" WIDTH="300" HEIGHT="150" SRC="house4.jpg">
 </IMAGE>

 <MLS>
               <MLS-CODE SECURITY="Restricted">
   29389877
  </MLS-CODE>
               <MLS-SOURCE SECURITY="Public">
   <NAME>
    Mary the Realtor
   </NAME>
   <PHONE>
    1-617-555-3333
   </PHONE>
   <FAX>
    1-617-555-4444
   </FAX>
   <WEB>
    <EMAIL>
      Mary@somebucks.com
    </EMAIL>
    <SITE>
      www.bigbucks.com
    </SITE>
   </WEB>
  </MLS-SOURCE>
 </MLS>

 <TYPE>
               Home
 </TYPE>

 <PRICE>
  $449,000
 </PRICE>

        <AGE UNITS="YEARS">
  7
```

```
</AGE>

       <LOCATION COUNTRY="USA" STATE="MA" COUNTY="MIDDLESEX" SECURITY="Public">
 <ADDRESS>
  100 Any Road
 </ADDRESS>
 <CITY>
     Lexington
 </CITY>
 <ZIP>
  02421
 </ZIP>
</LOCATION>

<STRUCTURE>
  <NUM-BEDS>
    7
          </NUM-BEDS>
  <NUM-BATHS>
    3
          </NUM-BATHS>
</STRUCTURE>

<DATES>
 <LISTING-DATE>
     6/8/98
                </LISTING-DATE>
</DATES>

       <LAND-AREA UNITS="ACRES">
 2.0
</LAND-AREA>

 </GENERAL>

 <FEATURES>
    <DISCLOSURES>
 In your dreams.
    </DISCLOSURES>
 <UTILITIES>
    Yes
 </UTILITIES>
 <EXTRAS>
    Pest control included.
 </EXTRAS>
 <CONSTRUCTION>
    Wallboard and glue
 </CONSTRUCTION>
 <ACCESS>
    Front door.
 </ACCESS>
 </FEATURES>

 <FINANCIAL>
 <ASSUMABLE>
    I assume so.
 </ASSUMABLE>
 <OWNER-CARRY>
    Too heavy.
 </OWNER-CARRY>
 <ASSESMENTS>
    $300,000
 </ASSESMENTS>
 <DUES>
    $100
 </DUES>
 <TAXES>
```

```
    $2,000
  </TAXES>
  <LENDER>
    Fly by nite mortgage co.
  </LENDER>
  <EARNEST>
    Burt
  </EARNEST>
  <DIRECTIONS>
    North, south, east, west
  </DIRECTIONS>
  </FINANCIAL>

  <REMARKS>
  </REMARKS>

  <CONTACTS>
 <COMPANY>
  <NAME>
    Noplace Realty
  </NAME>
  <ADDRESS>
    12 Main Street
  </ADDRESS>
  <CITY>
    Lowell, MA
  </CITY>
  <ZIP>
    34567
  </ZIP>
 </COMPANY>
 <AGENT>
  <NAME>
    Mary Jones
  </NAME>
  <ADDRESS>
  </ADDRESS>
  <CITY>
  </CITY>
  <ZIP>
  </ZIP>
 </AGENT>
 <OWNER>
  <NAME>
  </NAME>
  <ADDRESS>
  </ADDRESS>
  <CITY>
  </CITY>
  <ZIP>
  </ZIP>
 </OWNER>
    <TENANT>
    Yes.
 </TENANT>
    <COMMISION>
    15%
 </COMMISION>
  </CONTACTS>

</RESIDENTIAL-LISTING>

</RESIDENTIAL-LISTINGS>
```

Esempio: file XSL di esempio XSLReportProcessor

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
<?xml version="1.0"?>

<!-- Sample of styling an imagined real estate document. -->
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
       xmlns:fo="http://www.w3.org/1999/XSL/Format" >

  <xsl:template match="RESIDENTIAL-LISTINGS">
    <fo:root>
    <fo:layout-master-set>
 <fo:simple-page-master master-name="theMaster">
        <fo:region-body region-name="theRegion"/>
 </fo:simple-page-master>
  <fo:page-sequence-master master-name="theMaster">
    <fo:single-page-master-reference master-name="thePage" />
      </fo:page-sequence-master>
 </fo:layout-master-set>
  <fo:page-sequence master-name="theMaster">
  <fo:flow flow-name="theRegion">
          <xsl:apply-templates/>
 </fo:flow>
 </fo:page-sequence>
    </fo:root>
  </xsl:template>
  <xsl:template match="RESIDENTIAL-LISTING">


      <fo:block font-family="Times New Roman" font-weight="normal" font-size="24pt"
       background-color="silver" padding-before="5px" padding-after="5px"
       padding-start="5px" padding-end="5px" border-before-style="solid"
       border-before-color="blue" border-after-style="solid" border-after-color="blue"
       border-start-style="solid" border-start-color="blue" border-end-style="solid"
       border-end-color="blue">

 <fo:character character="y" background-color="blue" border-before-style="solid"
       border-before-color="yellow" border-after-style="solid" border-after-color="yellow"
       border-start-style="solid" border-start-color="yellow" border-end-style="solid"
       border-end-color="yellow" />
 </fo:block>


  </xsl:template>
</xsl:stylesheet>
```

Esempi: classi Resource

Questa sezione elenca gli esempi di codice forniti in tutta la documentazione delle classi Resource di IBM Toolbox per Java.

Resource e ChangeableResource

- Esempio: richiamo di un valore di attributo da RUser, una reale sottoclasse Resource
- Esempio: impostazione dei valori di attributo per RJob, una reale sottoclasse di ChangeableResource
- Esempio: utilizzo di un codice generico per accedere alle risorse

ResourceList

- Esempio: richiamo e stampa del contenuto di una ResourceList
- Esempio: utilizzo di un codice generico per accedere a una ResourceList
- Esempio: presentazione di un elenco di risorse in un servlet

Presentazione

- Esempio: utilizzo delle presentazioni

Il seguente esonero di responsabilità si applica a tutti gli esempi IBM Toolbox per Java:

Esonero di responsabilità per gli esempi di codice

IBM fornisce una licenza non esclusiva per utilizzare ciò come esempio da cui creare funzioni simili personalizzate, in base a richieste specifiche.

Tutti gli esempi di codice forniti dall'IBM hanno la sola funzione illustrativa. Questi esempi non sono stati interamente testati in tutte le condizioni. L'IBM, perciò, non intende implicita alcuna garanzia di affidabilità, manutenibilità o funzionalità di questi programmi.

Tutti i programmi qui contenuti sono forniti "COSI' COME SONO" senza garanzie di alcun tipo. Sono espressamente smentite tutte le garanzie implicite di non violazione, di commerciabilità e idoneità per scopi specifici.

Esempi: elenco di risorse

I seguenti esempi mostrano le varie modalità di gestione degli elenchi delle risorse:

- Esempio: richiamo e stampa del contenuto di una ResourceList
- Esempio: utilizzo di un codice generico per accedere ad una ResourceList
- Esempio: presentazione di un elenco di risorse in un servlet

Esempio: richiamo e stampa del contenuto di una ResourceList

Un esempio di una sottoclasse concreta di ResourceList è com.ibm.as400.resource.RJobList, che rappresenta un elenco di lavori iSeries. RJobList supporta molti ID selezione e ID ordinamento, ognuno dei quali può essere utilizzato per filtrare o ordinare l'elenco. Questo è un esempio per stampare il contenuto di una RJobList:

```
// Creare un oggetto RJobList per rappresentare un elenco di lavori.
AS400 system = new AS400("MYSYSTEM", "MYUSERID", "MYPASSWORD");
RJobList jobList = new RJobList(system);

// Filtrare l'elenco in modo da includere solo i lavori interattivi.
jobList.setSelectionValue(RJobList.JOB_TYPE, RJob.JOB_TYPE_INTERACTIVE);

// Ordinare l'elenco per nome utente, quindi per nome lavoro.
Object[] sortValue = new Object[] { RJob.USER_NAME, RJob.JOB_NAME };
jobList.setSortValue(sortValue);

// Aprire l'elenco e attenderne il completamento.
jobList.open();
jobList.waitForComplete();

// Leggere e stampare il contenuto dell'elenco.
long length = jobList.getListLength();
for(long i = 0; i < length; ++i)
{
        System.out.println(jobList.resourceAt(i));
}

// Chiudere l'elenco.
jobList.close();
```

Esonero di responsabilità per gli esempi di codice

Il seguente esonero di responsabilità si applica a tutti gli esempi IBM Toolbox per Java:

L'IBM fornisce una licenza non esclusiva per utilizzare tutti gli esempi del codice di programmazione da cui creare funzioni simili personalizzate, in base a richieste specifiche.

Questo codice di esempio è fornito dall'IBM con la sola funzione illustrativa. Questi esempi non sono stati interamente testati in tutte le condizioni. IBM, perciò, non fornisce nessun tipo di garanzia o affidabilità implicita, rispetto alla funzionalità o alle funzioni di questi programmi.

Tutti i programmi qui contenuti vengono forniti all'utente ″COSI' COME SONO″ senza garanzie di alcun tipo. Le garanzie implicite di non contraffazione, commerciabilità e adeguatezza a scopi specifici sono espressamente vietate.

Esempio: presentazione di un elenco di risorse in un servlet

Utilizzare la classe ResourceListRowData insieme alla classe HTMLFormConverter o HTMLTableConverter per presentare un elenco di risorse in un servlet.

- HTMLFormConverter visualizza il contenuto di un elenco di risorse come una serie di moduli, in cui ogni modulo contiene i valori dell'attributo per una risorsa dell'elenco.
- HTMLTableConverter visualizza il contenuto di un elenco di risorse come una tabella, in cui ogni riga contiene le informazioni riguardo una risorsa dell'elenco.

Le colonne per un oggetto ResourceListRowData vengono specificate come una schiera di ID dell'attributo colonna mentre ogni riga rappresenta un oggetto risorsa.

```
      // Creare l'elenco di risorse.
Questo esempio crea
      // tutti i messaggi presenti nella coda messaggi
      // dell'utente corrente.
      AS400 system = new AS400("MYSYSTEM", "MYUSERID", "MYPASSWORD");
      RMessageQueue messageQueue = new RMessageQueue(system, RMessageQueue.CURRENT);

   // Creare l'oggetto ResourceListRowData.  In questo esempio,
      // ci sono quattro colonne nella tabella. La prima colonna
   // contiene le icone e i nomi per ogni messaggio presente
      // nella coda messaggi. Le colonne rimanenti contengono testo,
   // severità e tipo per ogni messaggio.
      ResourceListRowData rowdata = new ResourceListRowData(messageQueue,
              new Object[] { null, RQueuedMessage.MESSAGE_TEXT, RQueuedMessage.MESSAGE_SEVERITY,
                                          RQueuedMessage.MESSAGE_TYPE } );

      // Creare gli oggetti HTMLTable e HTMLTableConverter da
      // usare per la creazione e la personalizzazione delle tabelle HTML.
            HTMLTable table = new HTMLTable();
      table.setCellSpacing(6);
      table.setBorderWidth(8);

      HTMLTableConverter converter = new HTMLTableConverter();
      converter.setTable(table);
      converter.setUseMetaData(true);

      // Generare la tabella HTML.
      String[] html = converter.convert(rowdata);
      System.out.println(html[0]);
```

Esempio: richiamo di un valore di attributo da una classe Resource

Una sottoclasse concreta di risorsa Resource com.ibm.as400.resource.RUser, che rappresento un utente iSeries.RUser supporta diversi ID attributo, ognuno dei quali è possibile utilizzare per richiamare valori di attributo.

L'esempio che segue richiama un valore di attributo da una RUser:

```
// Creare  un oggetto RUser che faccia riferimento ad uno specifico utente.
AS400 system = new AS400("MYSYSTEM", "MYUSERID", "MYPASSWORD");
RUser user = new RUser(system, "AUSERID");

// Richiamare il valore di attributo descrizione testo.
String textDescription = (String)user.getAttributeValue(RUser.TEXT_DESCRIPTION);
```

Esempio: impostazione dei valori di attributo per una ChangeableResource

Una sottoclasse concreta di ChangeableResource è com.ibm.as400.resource.RJob, che rappresenta un lavoro iSeries.RJob supporta molti ID attributo, ognuno dei quali può essere utilizzato per accedere ai valori di attributo.Questo esempio imposta due valori di attributo per un RJob:

```
// Creare un oggetto RJob per fare riferimento ad uno specifico lavoro.
AS400 system = new AS400("MYSYSTEM", "MYUSERID", "MYPASSWORD");
RJob job = new RJob(system, "AJOBNAME", "AUSERID", "AJOBNUMBER");

// Impostare il valore attributo del formato data.
job.setAttributeValue(RJob.DATE_FORMAT, RJob.DATE_FORMAT_JULIAN);

// Impostare il valore attributo ID regione o paese.
job.setAttributeValue(RJob.COUNTRY_ID, RJob.USER_PROFILE);

// Sincronizzare entrambe le modifiche attributo.
job.commitAttributeChanges();
```

Esempio: utilizzo di un codice generico per accedere alle risorse

E' possibile scrivere un codice generico per gestire qualsiasi sottoclasse Resource, ResourceList o ChangeableResource. Tale codice può migliorare la capacità di riutilizzo e di manutenzione e gestirà future sottoclassi Resource, ResourceList o ChangeableResource senza modifiche.

Ogni attributo ha un oggetto metadati dell'attributo associato (com.ibm.as400.resource.ResourceMetaData) che descrive le varie proprietà dell'attributo.Tali proprietà indicano anche se l'attributo è o meno di sola lettura e definiscono i valori predefiniti e i valori possibili.

Esempi:

Esempio: stampa del contenuto di ResourceList

Questo è l'esempio di un codice generico che stampa una parte del contenuto di una ResourceList:

```
    void printContents(ResourceList resourceList, long numberOfItems) throws ResourceException
  {
            // Aprire l'elenco e attendere che il numero richiesto di voci
            // sia disponibile.
            resourceList.open();
            resourceList.waitForResource(numberOfItems);

            for(long i = 0; i < numberOfItems; ++i)
     {
                    System.out.println(resourceList.resourceAt(i));
     }
  }
```

Esempio: utilizzo di ResourceMetaData per accedere ad ogni attributo supportato da una risorsa

Questo è un esempio di un codice generico che stampa il valore di ogni attributo supportato da una risorsa:

```
void printAllAttributeValues(Resource resource) throws ResourceException
{
        // Richiamare i metadati dell'attributo.
        ResourceMetaData[] attributeMetaData = resource.getAttributeMetaData();
```

```
        // Eseguire il loop di tutti gli attributi e stampare i valori.
        for(int i = 0; i < attributeMetaData.length; ++i)
    {
                    Object attributeID = attributeMetaData[i].getID();
                Object value = resource.getAttributeValue(attributeID);
                System.out.println("Attribute " + attributeID + " = " + value);
    }
}
```

Esempio: utilizzo di ResourceMetaData per reimpostare ogni attributo di una ChangeableResource

Questo è un esempio di codice generico che ripristina tutti gli attributi di ChangeableResource ai rispettivi valori predefiniti:

```
void resetAttributeValues(ChangeableResource resource) throws ResourceException
{
        // Richiamare i metadati dell'attributo.
        ResourceMetaData[] attributeMetaData = resource.getAttributeMetaData();

        // Eseguire il loop di tutti gli attributi.
        for(int i = 0; i < attributeMetaData.length; ++i)
    {
                // Se l'attributo è modificabile (non di sola lettura),
                // reimpostare il valore su quello predefinito.
                if (! attributeMetaData[i].isReadOnly())
{
                        Object attributeID = attributeMetaData[i].getID();
                        Object defaultValue = attributeMetaData[i].getDefaultValue();
                        resource.setAttributeValue(attributeID, defaultValue);
        }
    }

        // Sincronizzare tutte le modifiche attributo.
        resource.commitAttributeChanges();
}
```

Esempi: RFML

Questa sezione elenca gli esempi di codice forniti in tutta la documentazione del componente RFML di IBM Toolbox per Java:

- Esempio: utilizzo di RFML rispetto all'utilizzo delle classi Record di IBM Toolbox per Java
- Esempio: file sorgente RFML

Il seguente esonero di responsabilità si applica a tutti gli esempi IBM Toolbox per Java:

Esonero di responsabilità per gli esempi di codice

L'IBM fornisce una licenza non esclusiva per utilizzare tutti gli esempi del codice di programmazione da cui creare funzioni simili personalizzate, in base a richieste specifiche.

Questo codice di esempio è fornito dall'IBM con la sola funzione illustrativa. Questi esempi non sono stati interamente testati in tutte le condizioni. IBM, perciò, non fornisce nessun tipo di garanzia o affidabilità implicita, rispetto alla funzionalità o alle funzioni di questi programmi.

Tutti i programmi qui contenuti vengono forniti all'utente ″COSI' COME SONO″ senza garanzie di alcun tipo. Le garanzie implicite di non contraffazione, commerciabilità e adeguatezza a scopi specifici sono espressamente vietate.

Esempio: file sorgente RFML

Questo file sorgente RFML di esempio definisce il formato dei record del cliente come è stato utilizzato nell'esempio RFML Utilizzare RFML rispetto ad utilizzare le classi record di Toolbox per Java. Questo file sorgente RFML potrebbe essere un file di testo denominato qcustcdt.rfml.

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE rfml SYSTEM "rfml.dtd">

<rfml version="4.0" ccsid="819">

  <recordformat name="cusrec">

    <data name="cusnum" type="zoned" length="6"  precision="0" init="0"/>
    <data name="lstnam" type="char"  length="8"  ccsid="37" init="A"/>
    <data name="init"   type="char"  length="3"  ccsid="37" init="B"/>
    <data name="street" type="char"  length="13" ccsid="37" init="C"/>
    <data name="city"   type="char"  length="6"  ccsid="37" init="D"/>
    <data name="state"  type="char"  length="2"  ccsid="37" init="E"/>
    <data name="zipcod" type="zoned" length="5" init="1"/>
    <data name="cdtlmt" type="zoned" length="4" init="2"/>
    <data name="chgcod" type="zoned" length="1" init="3"/>
    <data name="baldue" type="zoned" length="6" precision="2" init="4"/>
    <data name="cdtdue" type="zoned" length="6" precision="2" init="5"/>

  </recordformat>

  <recordformat name="cusrec1">

    <data name="cusnum" type="zoned" length="6"  precision="0" init="0"/>
    <data name="lstnam" type="char"  length="8"  ccsid="37" init="A"/>
    <data name="init"   type="char"  length="3"  ccsid="37" init="B"/>
    <data name="street" type="char"  length="13" ccsid="37" init="C"/>
    <data name="city"   type="char"  length="6"  ccsid="37" init="D"/>
    <data name="state"  type="char"  length="2"  ccsid="37" init="E"/>
    <data name="zipcod" type="zoned" length="5" init="1"/>
    <data name="cdtlmt" type="zoned" length="4" init="2"/>
    <data name="chgcod" type="zoned" length="1" init="3"/>
    <data name="baldue" type="struct" struct="balance"/>
    <data name="cdtdue" type="struct" struct="balance"/>

  </recordformat>

  <recordformat name="cusrecAscii">

    <data name="cusnum" type="zoned" length="6"  precision="0" init="0"/>
    <data name="lstnam" type="char"  length="8" init="A"/>
    <data name="init"   type="char"  length="3" init="B"/>
    <data name="street" type="char"  length="13" init="C"/>
    <data name="city"   type="char"  length="6" init="D"/>
    <data name="state"  type="char"  length="2" init="E"/>
    <data name="zipcod" type="zoned" length="5" init="1"/>
    <data name="cdtlmt" type="zoned" length="4" init="2"/>
    <data name="chgcod" type="zoned" length="1" init="3"/>
    <data name="baldue" type="zoned" length="6" precision="2" init="4"/>
    <data name="cdtdue" type="zoned" length="6" precision="2" init="5"/>

  </recordformat>

  <struct name="balance">
    <data name="amount" type="zoned" length="6" precision="2" init="7"/>
  </struct>

</rfml>
```

Esempio: utilizzo di una credenziale di token di profilo per scambiare l'identità del sottoprocesso OS/400

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

L'esempio di codice che segue mostra come utilizzare una credenziale token di profilo per scambiare l'identità del sottoprocesso OS/400 ed eseguire le funzioni per conto di un utente specifico:

```
        // Prepararsi a gestire il sistema AS/400 locale.
        AS400 system = new AS400("localhost", "*CURRENT", "*CURRENT");

        // Creare una ProfileTokenCredential ad utilizzo singolo con un supero tempo di 60 secondi.
        // Devono essere sostituiti un ID utente e una parola d'ordine validi.
        ProfileTokenCredential pt = new ProfileTokenCredential();
        pt.setSystem(system);
        pt.setTimeoutInterval(60);
        pt.setTokenType(ProfileTokenCredential.TYPE_SINGLE_USE);

    pt.setTokenExtended("USERID", "PASSWORD");

        // Scambiare l'identità del sottoprocesso OS/400, richiamando una credenziale per
        // tornare successivamente all'identità originale.
        AS400Credential cr = pt.swap(true);

        // Eseguire il lavoro a questo punto sotto l'identità scambiata.

        // Tornare all'identità originale del sottoprocesso OS/400.
        cr.swap();

        // Eliminare il contenuto delle credenziali.
        cr.destroy();
        pt.destroy();
```

Esempi dalle classi servlet

Gli esempi che seguono mostrano le varie modalità di utilizzo delle classi servlet:

- Esempio: utilizzo della classe ListRowData
- Esempio: utilizzo della classe RecordListRowData
- Esempio: utilizzo della classe SQLResultSetRowData
- Esempio: utilizzo della classe HTMLFormConverter
- Esempio: utilizzo della classe ListMetaData
- Esempio: utilizzo della classe SQLResultSetMetaData
- Esempio: presentazione di un elenco di risorse in un servlet

E' inoltre possibile utilizzare il servlet e le classi HTML insieme, come in questo esempio.

Esonero di responsabilità per gli esempi di codice

Il seguente esonero di responsabilità si applica a tutti gli esempi IBM Toolbox per Java:

> L'IBM fornisce una licenza non esclusiva per utilizzare tutti gli esempi del codice di programmazione da cui creare funzioni simili personalizzate, in base a richieste specifiche.
>
> Questo codice di esempio è fornito dall'IBM con la sola funzione illustrativa. Questi esempi non sono stati interamente testati in tutte le condizioni. IBM, perciò, non fornisce nessun tipo di garanzia o affidabilità implicita, rispetto alla funzionalità o alle funzioni di questi programmi.

Tutti i programmi qui contenuti vengono forniti all'utente ″COSI' COME SONO″ senza garanzie di alcun tipo. Le garanzie implicite di non contraffazione, commerciabilità e adeguatezza a scopi specifici sono espressamente vietate.

Esempio: utilizzo di ListRowData

Questo esempio è composto da tre parti:

- "Sorgente Java che illustra il funzionamento della classe ListRowData"
- "Sorgente HTML creato dal sorgente Java utilizzando HTMLTableConverter"
- "Come un browser visualizza l'HTML creato" a pagina 648

Sorgente Java che illustra il funzionamento della classe ListRowData

```
         // Accedere ad una coda dati non vuota esistente
KeyedDataQueue dq = new KeyedDataQueue(systemObject_, "/QSYS.LIB/MYLIB.LIB/MYDQ.DTAQ");

         // Creare un oggetto metadati.
         ListMetaData metaData = new ListMetaData(2);

         // Impostare la prima colonna che deve essere l'ID cliente.
metaData.setColumnName(0, "Customer ID");
metaData.setColumnLabel(0, "Customer ID");
         metaData.setColumnType(0, RowMetaDataType.STRING_DATA_TYPE);

         // Impostare la seconda colonna che deve essere l'ordine da elaborare.
metaData.setColumnName(1, "Order Number");
metaData.setColumnLabel(1, "Order Number");
         metaData.setColumnType(1, RowMetaDataType.STRING_DATA_TYPE);

         // Creare un oggetto ListRowData.
         ListRowData rowData = new ListRowData();
         rowData.setMetaData(metaData);

         // Richiamare le voci dalla coda dati.
KeyedDataQueueEntry data = dq.read(key, 0, "EQ");
            while (data != null)
{
               // Aggiungere la voce della coda nell'oggetto dati di riga.
               Object[] row = new Object[2];
               row[0] = new String(key);
               row[1] = new String(data.getData());
               rowData.addRow(row);

               // Richiamare un'altra voce dalla coda.
   data = dq.read(key, 0, "EQ");
}

         // Creare un oggetto programma di conversione HTML e convertire il rowData in HTML.
         HTMLTableConverter conv = new HTMLTableConverter();
         conv.setUseMetaData(true);
         HTMLTable[] html = conv.convertToTables(rowData);

         // Visualizzare l'emissione del programma di conversione.
         System.out.println(html[0]);
```

Sorgente HTML creato dal sorgente Java utilizzando HTMLTableConverter

L'utilizzo di "Classe HTMLTableConverter" a pagina 242 nell'esempio del sorgente Java precedente crea il seguente codice HTML.

```
<table>
<tr>
<th>Customer ID</th>
<th>Order Number</th>
```

```
</tr>
<tr>
<td>777-53-4444</td>
<td>12345-XYZ</td>
</tr>
<tr>
<td>777-53-4444</td>
<td>56789-ABC</td>
</tr>
</table>
```

Come un browser visualizza l'HTML creato

La seguente tabella mostra come appare il codice sorgente HTML quando viene visualizzato in un browser.

ID cliente	Numero di ordine
777-53-4444	12345-XYZ
777-53-4444	56789-ABC

Esempio: utilizzo di RecordListRowData

Questo esempio è composto da tre parti:

- Sorgente Java che mostra come funziona la classe RecordListRowData
- Sorgente HTML generato dal sorgente Java tramite HTMLTableConverter
- Modalità di visualizzazione dell'HTML creato da parte del browser

Sorgente Java che mostra il funzionamento della classe RecordListRowData

```
            // Creare un oggetto server.
        AS400 mySystem = new AS400 ("mySystem.myComp.com", "UserId", "Password");

            // Richiamare il nome percorso per il file.
        QSYSObjectPathName file = new QSYSObjectPathName(myLibrary, myFile, "%first%", "mbr");
            String ifspath = file.getPath();

            // Creare un oggetto file che rappresenta il file.

            SequentialFile sf = new SequentialFile(mySystem, ifspath);

            // Reperire il formato record dal file.
         AS400FileRecordDescription recordDescription = new AS400FileRecordDescription(mySystem, ifspath);
            RecordFormat recordFormat = recordDescription.retrieveRecordFormat()[0];

            // Impostare il formato record per il file.
            sf.setRecordFormat(recordFormat);

            // Richiamare i record nel file.
            Record[] records = sf.readAll();

            // Creare un oggetto RecordListRowData e aggiungere i record.
            RecordListRowData rowData = new RecordListRowData(recordFormat);

            for (int i=0; i < records.length; i++)
        {
                  rowData.addRow(records[i]);
        }

            // Creare un oggetto programma di conversione HTML e convertire il rowData in HTML.
            HTMLTableConverter conv = new HTMLTableConverter();
            conv.setMaximumTableSize(3);
```

```
HTMLTable[] html = conv.convertToTables(rowData);

// Visualizzare la prima tabella HTML generata dal programma di conversione.
System.out.println(html[0]);
```

Sorgente HTML generato dal sorgente Java utilizzando HTMLTableConverter

L'utilizzo della classe HTMLTableConverter nell'esempio di sorgente Java precedentemente riportato crea il seguente codice HTML.

```
<table>
<tr>
<th>CNUM</th>
<th>LNAM</th>
<th>INIT</th>
<th>STR</th>
<th>CTY</th>
<th>STATE</th>
<th>ZIP</th>
<th>CTLMT</th>
<th>CHGCOD</th>
<th>BDUE</th>
<th>CTDUE</th>
</tr>
<tr>
<td>938472</td>
<td>Henning </td>
<td>G K</td>
<td>4859 Elm Ave </td>
<td>Dallas</td>
<td>TX</td>
<td align="right">75217</td>
<td align="right">5000</td>
<td align="right">3</td>
<td align="right">37.00</td>
<td align="right">0.00</td>
</tr>
<tr>
<td>839283</td>
<td>Jones   </td>
<td>B D</td>
<td>21B NW 135 St</td>
<td>Clay  </td>
<td>NY</td>
<td align="right">13041</td>
<td align="right">400</td>
<td align="right">1</td>
<td align="right">100.00</td>
<td align="right">0.00</td>
</tr>
<tr>
<td>392859</td>
<td>Vine    </td>
<td>S S</td>
<td>PO Box 79    </td>
<td>Broton</td>
<td>VT</td>
<td align="right">5046</td>
<td align="right">700</td>
<td align="right">1</td>
<td align="right">439.00</td>
<td align="right">0.00</td>
</tr>
</table>
```

Modalità di visualizzazione dell'HTML creato da parte del browser

La seguente tabella mostra come appare il codice sorgente HTML quando viene visualizzato in un browser.

CNUM	LNAM	INIT	STR	CTY	STATE	ZIP	CTLMT	CHGCOD	BDUE	CTDUE
938472	Henning	G K	4859 Elm Ave	Dallas	TX	75217	5000	3	37.00	0.00
839283	Jones	B D	21B NW 135 St	Clay	NY	13041	400	1	100.00	0.00
392859	Vine	S S	PO Box 79	Broton	VT	5046	700	1	439.00	0.00

Esempio: utilizzo di SQLResultSetRowData

Questo esempio è composto da tre parti:

- Sorgente Java che mostra come funziona la classe SQLResultSetRowData
- Sorgente HTML generato dal sorgente Java tramite HTMLTableConverter
- Modalità di visualizzazione dell'HTML creato da parte del browser

Sorgente Java che mostra il funzionamento della classe SQLResultSetRowData

```
            // Creare un oggetto server.
        AS400 mySystem = new AS400 ("mySystem.myComp.com", "UserId", "Password");

            // Registrare e ottenere un collegamento al database.
               DriverManager.registerDriver(new com.ibm.as400.access.AS400JDBCDriver());
        Connection connection = DriverManager.getConnection("jdbc:as400://" + mySystem.getSystemName());

            // Eseguire un'istruzione SQL e richiamare la serie di risultati.
            Statement statement = connection.createStatement();
        statement.execute("select * from qiws.qcustcdt");
            ResultSet resultSet = statement.getResultSet();

            // Creare l'oggetto SQLResultSetRowData e inizializzarlo nella serie di risultati.
            SQLResultSetRowData rowData = new SQLResultSetRowData(resultSet);

            // Creare un oggetto tabella HTML che deve essere usato dal programma di conversione.
            HTMLTable table = new HTMLTable();

            // Impostare le intestazioni della colonna descrittiva.
        String[] headers = {"Customer Number", "Last Name", "Initials",
                                           "Street Address", "City", "State", "Zip Code",
                                           "Credit Limit", "Charge Code", "Balance Due",
                                           "Credit Due"};
            table.setHeader(headers);

            // Impostare diverse opzioni di formattazione nella tabella.
            table.setBorderWidth(2);
            table.setCellSpacing(1);
            table.setCellPadding(1);

            // Creare un oggetto programma di conversione HTML e convertire il rowData in HTML.
            HTMLTableConverter conv = new HTMLTableConverter();
            conv.setTable(table);
            HTMLTable[] html = conv.convertToTables(rowData);

            // Visualizzare la tabella HTML generata dal programma di conversione.
          System.out.println(html[0]);
```

Sorgente HTML generato dal sorgente Java utilizzando HTMLTableConverter

L'utilizzo della classe HTMLTableConverter nell'esempio di sorgente Java precedentemente riportato crea il seguente codice HTML.

```
<table border="2" cellpadding="1" cellspacing="1">
<tr>
<th>Customer Number</th>
<th>Last Name</th>
```

```
<th>Initials</th>
<th>Street Address</th>
<th>City</th>
<th>State</th>
<th>Zip Code</th>
<th>Credit Limit</th>
<th>Charge Code</th>
<th>Balance Due</th>
<th>Credit Due</th>
</tr>
<tr>
<td>938472</td>
<td>Henning </td>
<td>G K</td>
<td>4859 Elm Ave </td>
<td>Dallas</td>
<td>TX</td>
<td align="right">75217</td>
<td align="right">5000</td>
<td align="right">3</td>
<td align="right">37.00</td>
<td align="right">0.00</td>
</tr>
<tr>
<td>839283</td>
<td>Jones   </td>
<td
>B D</td>
<td>21B NW 135 St</td>
<td>Clay  </td>
<td>NY</td>
<td align="right">13041</td>
<td align="right">400</td>
<td align="right">1</td>
<td align="right">100.00</td>
<td align="right">0.00</td>
</tr>
<tr>
<td>392859</td>
<td>Vine    </td>
<td>S S</td>
<td>PO Box 79    </td>
<td>Broton</td>
<td>VT</td>
<td align="right">5046</td>
<td align="right">700</td>
<td align="right">1</td>
<td align="right">439.00</td>
<td align="right">0.00</td>
</tr>
<tr>
<td>938485</td>
<td>Johnson </td>
<td>J A</td>
<td>3 Alpine Way </td>
<td>Helen </td>
<td>GA</td>
<td align="right">30545</td>
<td align="right">9999</td>
<td align="right">2</td>
<td align="right">3987.50</td>
<td align="right">33.50</td>
</tr>
<tr>
<td>397267</td>
<td>Tyron   </td>
<td>W E</td>
```

```
<td>13 Myrtle Dr </td>
<td>Hector</td>
<td>NY</td>
<td align="right">14841</td>
<td align="right">1000</td>
<td align="right">1</td>
<td align="right">0.00</td>
<td align="right">0.00</td>
</tr>
<tr>
<td>389572</td>
<td>Stevens </td>
<td>K L</td>
<td>208 Snow Pass</td>
<td>Denver</td>
<td>CO</td>
<td align="right">80226</td>
<td align="right">400</td>
<td align="right">1</td>
<td align="right">58.75</td>
<td align="right">1.50</td>
</tr>
<tr>
<td>846283</td>
<td>Alison  </td>
<td>J S</td>
<td>787 Lake Dr  </td>
<td>Isle  </td>
<td>MN</td>
<td align="right">56342</td>
<td align="right">5000</td>
<td align="right">3</td>
<td align="right">10.00</td>
<td align="right">0.00</td>
</tr>
<tr>
<td>475938</td>
<td>Doe     </td>
<td>J W</td>
<td>59 Archer Rd </td>
<td>Sutter</td>
<td>CA</td>
<td align="right">95685</td>
<td align="right">700</td>
<td align="right">2</td>
<td align="right">250.00</td>
<td align="right">100.00</td>
</tr>
<tr>
<td>693829</td>
<td>Thomas  </td>
<td>A N</td>
<td>3 Dove Circle</td>
<td>Casper</td>
<td>WY</td>
<td align="right">82609</td>
<td align="right">9999</td>
<td align="right">2</td>
<td align="right">0.00</td>
<td align="right">0.00</td>
</tr>
<tr>
<td>593029</td>
<td>Williams</td>
<td>E D</td>
<td>485 SE 2 Ave </td>
<td>Dallas</td>
```

```
<td>TX</td>
<td align="right">75218</td>
<td align="right">200</td>
<td align="right">1</td>
<td align="right">25.00</td>
<td align="right">0.00</td>
</tr>
<tr>
<td>192837</td>
<td>Lee     </td>
<td>F L</td>
<td>5963 Oak St  </td>
<td>Hector</td>
<td>NY</td>
<td align="right">14841</td>
<td align="right">700</td>
<td align="right">2</td>
<td align="right">489.50</td>
<td align="right">0.50</td>
</tr>
<tr>
<td>583990</td>
<td>Abraham </td>
<td>M T</td>
<td>392 Mill St  </td>
<td>Isle  </td>
<td>MN</td>
<td align="right">56342</td>
<td align="right">9999</td>
<td align="right">3</td>
<td align="right">500.00</td>
<td align="right">0.00</td>
</tr>
</table>
```

Modalità di visualizzazione dell'HTML creato da parte del browser

La seguente tabella mostra come appare il codice sorgente HTML quando viene visualizzato in un browser.

Customer Number	Last Name	Initials	Street Address	City	State	Zip Code	Credit Limit	Charge Code	Balance Due	Credit Due
938472	Henning	G K	4859 Elm Ave	Dallas	TX	75217	5000	3	37.00	0.00
839283	Jones	B D	21B NW 135 St	Clay	NY	13041	400	1	100.00	0.00
392859	Vine	S S	PO Box 79	Broton	VT	5046	700	1	439.00	0.00
938485	Johnson	J A	3 Alpine Way	Helen	GA	30545	9999	2	3987.50	33.50
397267	Tyron	W E	13 Myrtle Dr	Hector	NY	14841	1000	1	0.00	0.00
389572	Stevens	K L	208 Snow Pass	Denver	CO	80226	400	1	58.75	1.50
846283	Alison	J S	787 Lake Dr	Isle	MN	56342	5000	3	10.00	0.00
475938	Doe	J W	59 Archer Rd	Sutter	CA	95685	700	2	250.00	100.00
693829	Thomas	A N	3 Dove Circle	Casper	WY	82609	9999	2	0.00	0.00

Customer Number	Last Name	Initials	Street Address	City	State	Zip Code	Credit Limit	Charge Code	Balance Due	Credit Due
593029	Williams	E D	485 SE 2 Ave	Dallas	TX	75218	200	1	25.00	0.00
192837	Lee	F L	5963 Oak St	Hector	NY	14841	700	2	489.50	0.50
583990	Abraham	M T	392 Mill St	Isle	MN	56342	9999	3	500.00	0.00

Esempio: utilizzo di HTMLFormConverter

Mentre si avvia un server web con un supporto servlet, compilare ed eseguire gli esempi seguenti, per comprendere il funzionamento di HTMLFormConverter:

```
import java.awt.Color;
import java.io.IOException;
import java.io.PrintWriter;
import java.sql.DriverManager;
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.Statement;
import java.util.Enumeration;
import java.util.Hashtable;
import javax.servlet.*;
import javax.servlet.http.*;

import com.ibm.as400.util.html.GridLayoutFormPanel;
import com.ibm.as400.util.html.HTMLConstants;
import com.ibm.as400.util.html.HTMLForm;
import com.ibm.as400.util.html.HTMLTable;
import com.ibm.as400.util.html.HTMLTableCaption;
import com.ibm.as400.util.html.HTMLText;
import com.ibm.as400.util.html.LabelFormElement;
import com.ibm.as400.util.html.LineLayoutFormPanel;
import com.ibm.as400.util.html.SubmitFormInput;
import com.ibm.as400.util.html.TextFormInput;

import com.ibm.as400.util.servlet.HTMLFormConverter;
import com.ibm.as400.util.servlet.SQLResultSetRowData;

import com.ibm.as400.access.AS400;
import com.ibm.as400.access.AS400JDBCDriver;

/**
 * Un esempio di utilizzo della classe HTMLFormConverter in un servlet.
 */
public class HTMLFormConverterExample extends HttpServlet

{
   private String userId_   = "myUserId";
   private String password_ = "myPwd";
   private AS400 system_;

   private Connection databaseConnection_;



   // Eseguire la ripulitura prima di tornare al modulo HTML principale.
   public void cleanup()
   {
      try
      {
         // Chiudere il collegamento al database.
         if (databaseConnection_ != null)
```

```
      {
         databaseConnection_.close();
         databaseConnection_ = null;
      }
   }
   catch (Exception e)
   {
      e.printStackTrace ();
   }
}


// Convertire i dati di riga nell'HTML formattato.
private HTMLTable[] convertRowData(SQLResultSetRowData rowData)
{
   try
   {
      // Creare il programma di conversione, che genererà HTML dalla
      // serie di risultati restituita dall'interrogazione database.
      HTMLFormConverter converter = new HTMLFormConverter();

      // Impostare gli attributi del modulo.
               converter.setBorderWidth(3);
               converter.setCellPadding(2);
               converter.setCellSpacing(4);

      // Convertire i dati riga in HTML.
               HTMLTable[] htmlTable = converter.convertToForms(rowData);
               return htmlTable;
   }
   catch (Exception e)
   {
      e.printStackTrace ();
      return null;
   }

}


// Restituire la risposta al client.
public void doGet(HttpServletRequest request, HttpServletResponse response)
   throws IOException, ServletException
{
   response.setContentType("text/html");
   PrintWriter out = response.getWriter();

   out.println(showHtmlMain());
   out.close();
}


// Gestire i dati inviati al modulo.
public void doPost (HttpServletRequest request, HttpServletResponse response)
   throws ServletException, IOException
{
   SQLResultSetRowData rowData = new SQLResultSetRowData();
   HTMLTable[] htmlTable = null;

   // Richiamare l'oggetto sessione corrente o crearne uno se necessario.
   HttpSession session = request.getSession(true);

   ServletOutputStream out = response.getOutputStream();

   response.setContentType("text/html");

   Hashtable parameters = getRequestParameters (request);
```

```
// Reperire i valori dati riga ed i valori di tabella HTML per questa sessione.
rowData = (SQLResultSetRowData) session.getValue("sessionRowData");
htmlTable = (HTMLTable[]) session.getValue("sessionHtmlTable");

// se questa è la prima volta, visualizzare il primo record
if (parameters.containsKey("getRecords"))
{
   rowData = getAllRecords(parameters, out);

   if (rowData != null)
   {
      // Impostare il valore dati riga per questa sessione.
      session.putValue("sessionRowData", rowData);

      // Posizionarsi sul primo record.
      rowData.first();

      // Convertire i dati di riga nell'HTML formattato.
      htmlTable = convertRowData(rowData);

      if (htmlTable != null)
      {
         rowData.first();
         session.putValue("sessionHtmlTable", htmlTable);
         out.println(showHtmlForRecord(htmlTable, 0));
      }
   }
}
// Se è stato premuto il pulsante "Torna al principale",
// tornare al modulo HTML principale
else if (parameters.containsKey("returnToMain"))
{
   session.invalidate();
   cleanup();
   out.println(showHtmlMain());
}
// se è stato premuto il pulsate "Primo", viene visualizzato il primo record
else if (parameters.containsKey("getFirstRecord"))
{
   rowData.first();
   out.println(showHtmlForRecord(htmlTable, 0));
}
// se è stato premuto il pulsante "Precedente", viene visualizzato il record precedente
else if (parameters.containsKey("getPreviousRecord"))
{
   if (!rowData.previous())
   {
      rowData.first();
   }
   out.println(showHtmlForRecord(htmlTable, rowData.getCurrentPosition()));
}
// se è stato premuto il pulsate "Successivo", viene visualizzato il record successivo
else if (parameters.containsKey("getNextRecord"))
{
   if (!rowData.next())
   {
      rowData.last();
   }
   out.println(showHtmlForRecord(htmlTable, rowData.getCurrentPosition()));
}
// se è stato premuto il pulsante "Ultimo", viene visualizzato l'ultimo record
else if (parameters.containsKey("getLastRecord"))
{
   rowData.last();
   out.println(showHtmlForRecord(htmlTable, rowData.getCurrentPosition()));
}
// se non si è verifica una delle condizioni precedenti, deve esservi stato un errore
```

```
   else
   {
      out.println(showHtmlForError("Internal error occurred. Unexpected parameters."));
   }

   // Salvare il valore dati riga per questa sessione in modo che la posizione corrente
   // venga aggiornata nell'oggetto associato alla sessione.
   session.putValue("sessionRowData", rowData);

   // Chiudere il flusso di emissione
   out.close();
}


// Richiamare tutti i record dall'immissione file da parte dell'utente.
private SQLResultSetRowData getAllRecords(Hashtable parameters, ServletOutputStream out)
   throws IOException
{
   SQLResultSetRowData records = null;

   try
   {
      // Richiamare il nome sistema, libreria e file dall'elenco di parametri.
      String sys = ((String) parameters.get("System")).toUpperCase();
      String lib = ((String) parameters.get("Library")).toUpperCase();
      String file = ((String) parameters.get("File")).toUpperCase();
      if ((sys == null || sys.equals("")) ||
          (lib == null || lib.equals("")) ||
          (file == null || file.equals("")))
      {
         out.println(showHtmlForError("Invalid system, file or library name."));
      }
      else
      {
         // Richiamare il collegamento al server.
         getDatabaseConnection (sys, out);
         if (databaseConnection_ != null)
         {
            Statement sqlStatement = databaseConnection_.createStatement();

            // Interrogare il database per ottenere la serie di risultati.
            String query = "SELECT * FROM " + lib + "." + file;
            ResultSet rs = sqlStatement.executeQuery (query);

            boolean rsHasRows = rs.next();  // posizionare il cursore sulla prima riga

            // Visualizzare un messaggio di errore se il file non contiene record;
            // altrimenti, impostare i dati riga sui dati serie di risultati
            if (!rsHasRows)
            {
               out.println(showHtmlForError("No records in the file."));
            }
            else
            {
               records = new SQLResultSetRowData (rs);
            }

            // Non chiudere Statement prima di aver terminato di utilizzare
            // ResultSet o potrebbero verificarsi dei problemi.
            sqlStatement.close();
         }
      }
   }
   catch (Exception e)
   {
      e.printStackTrace ();
      out.println(showHtmlForError(e.toString()));
```

```
   }

   return records;
}


// Stabilire un collegamento al database.
private void getDatabaseConnection (String sysName, ServletOutputStream out)
   throws IOException
{
   if (databaseConnection_ == null)
   {
      try
      {
         databaseConnection_ =
            DriverManager.getConnection("jdbc:as400://sysName,userId_,password_ );
      }
      catch (Exception e)
      {
         e.printStackTrace ();
         out.println(showHtmlForError(e.toString()));
      }
   }
}


// Richiama i parametri da una richiesta HTTP servlet.
private static Hashtable getRequestParameters (HttpServletRequest request)
{
   Hashtable parameters = new Hashtable ();
   Enumeration enum = request.getParameterNames();
   while (enum.hasMoreElements())
   {
      String key = (String) enum.nextElement();
      String value = request.getParameter (key);
      parameters.put (key, value);
   }
   return parameters;
}


// Richiamare le informazioni sul servlet.
public String getServletInfo()
{
   return "HTMLFormConverterExample";
}


// Eseguire le fasi dell'inizializzazione.
public void init(ServletConfig config)
{
   try
   {
      super.init(config);

      // Registrare l'unità di controllo JDBC
      try
      {
         DriverManager.registerDriver(new AS400JDBCDriver());
      }
      catch (Exception e)
      {
         System.out.println("JDBC Driver not found");
      }
   }
   catch (Exception e)
   {
```

```
      e.printStackTrace();
   }
}


// Impostare le info sull'intestazione pagina.
private String showHeader(String title)
{
   StringBuffer page = new StringBuffer();
   page.append("<html><head><title>" + title + "</title>");
   page.append("</head><body bgcolor=\"blanchedalmond\">");
   return page.toString ();
}


// Visualizzare la pagina HTML con le informazioni sull'errore appropriate.
private String showHtmlForError(String message)
{
   String title = "Error";
   StringBuffer page = new StringBuffer();
   page.append (showHeader (title));
   try
   {
      // Creare l'oggetto modulo HTML
      HTMLForm errorForm = new HTMLForm("HTMLFormConverterExample");

      // Impostare in modo che doPost() venga chiamato quando si inoltra il modulo.
      errorForm.setMethod(HTMLForm.METHOD_POST);

      // Creare un pannello a colonna singola a cui verranno aggiunti gli
      // elementi HTML.
      GridLayoutFormPanel grid = new GridLayoutFormPanel();

      // Creare l'elemento testo per l'errore ed aggiungerlo al pannello.
      HTMLText text = new HTMLText(message);
      text.setBold(true);
      text.setColor(Color.red);
      grid.addElement(text);

      // Creare il pulsante per tornare al principale ed aggiungerlo al pannello.
      grid.addElement(new SubmitFormInput("returnToMain", "Return to Main"));

      // Aggiungere il pannello al modulo HTML.
      errorForm.addElement(grid);

      page.append(errorForm.toString());
   }
   catch (Exception e)
   {
      e.printStackTrace ();
   }
   page.append("</body></html>");
   return page.toString();
}


// Visualizzare il modulo HTML per un singolo record.
private String showHtmlForRecord(HTMLTable[] htmlTable, int position)
{
   String title = "HTMLFormConverter Example";
   StringBuffer page = new StringBuffer();
   page.append (showHeader (title));

   page.append("<h1>" + title + "</h1>");

   try
   {
```

```
      // Creare l'oggetto modulo HTML
      HTMLForm recForm = new HTMLForm("HTMLFormConverterExample");

      // Impostare in modo che doPost() venga chiamato quando si inoltra il modulo.
      recForm.setMethod(HTMLForm.METHOD_POST);

      // Impostare il layout di un pannello a colonna singola, nel quale disporre
      // gli elementi HTML generati.
      GridLayoutFormPanel grid = new GridLayoutFormPanel();

      // Creare e aggiungere un titolo tabella che tenga traccia
      // del record corrente.
      HTMLText recNumText = new HTMLText("Record number: " + (position + 1));
      recNumText.setBold(true);
      grid.addElement(recNumText);

      // Impostare il layout di un pannello a due colonne, nel quale disporre
      // la tabella ed il testo per commento sull'emissione del programma di conversione.
      GridLayoutFormPanel tableGrid = new GridLayoutFormPanel(2);
      tableGrid.addElement(htmlTable[position]);
      HTMLText comment = new HTMLText("   <---- Output from the HTMLFormConverter class");
      comment.setBold(true);
      comment.setColor(Color.blue);
      tableGrid.addElement(comment);

      // Aggiungere la riga di tabella al pannello.
      grid.addElement(tableGrid);

      // Impostare il layout di un pannello a riga singola, nel quale disporre
      // i pulsanti per spostarsi attraverso l'elenco di record.
      LineLayoutFormPanel buttonLine = new LineLayoutFormPanel();
      buttonLine.addElement(new SubmitFormInput("getFirstRecord", "First"));
      buttonLine.addElement(new SubmitFormInput("getPreviousRecord", "Previous"));
      buttonLine.addElement(new SubmitFormInput("getNextRecord", "Next"));
      buttonLine.addElement(new SubmitFormInput("getLastRecord", "Last"));

      // Impostare un altro layout di pannello a riga singola per il
      // pulsante Torna a Return a principale.
      LineLayoutFormPanel returnToMainLine = new LineLayoutFormPanel();
      returnToMainLine.addElement(new SubmitFormInput("returnToMain", "Return to Main"));

      // Aggiungere righe contenenti i pulsanti al pannello griglia.
      grid.addElement(buttonLine);
      grid.addElement(returnToMainLine);

      // Aggiungere il pannello al modulo.
      recForm.addElement(grid);

      // Aggiungere il modulo alla pagina HTML.
      page.append(recForm.toString());
   }
   catch (Exception e)
   {
      e.printStackTrace ();
   }

   page.append("</body></html>");
   return page.toString();
}


// Visualizzare il modulo HTML principale (richiedere immissione per il nome sistema, file e
// libreria).
private String showHtmlMain()
{
   String title = "HTMLFormConverter Example";
   StringBuffer page = new StringBuffer();
```

```
page.append (showHeader (title));

page.append("<h1>" + title + "</h1>");

// Creare l'oggetto modulo HTML
HTMLForm mainForm = new HTMLForm("HTMLFormConverterExample");

try
{
   // Impostare in modo che doPost() venga chiamato quando si inoltra il modulo.
   mainForm.setMethod(HTMLForm.METHOD_POST);

   // Aggiungere una breve descrizione al modulo.
   HTMLText desc =
      new HTMLText("<P>This example uses the HTMLFormConverter class " +
                   "to convert data retrieved from a server " +
                   "file.  The converter produces an array of HTML " +
                   "tables.  Each entry in the array is a record from " +
                   "the file.  " +
                   "Records are displayed one at a time, " +
                   "giving you buttons to move forward or backward " +
                   "through the list of records.</P>");
   mainForm.addElement(desc);

   // Aggiungere istruzioni al modulo.
   HTMLText instr =
      new HTMLText("<P>Please input the name of the server, " +
                   "and the file and library name for the file you " +
                   "wish to access.  Then push the Show Records " +
                   "button to continue.</P>");
   mainForm.addElement(instr);


   // Creare un pannello layout di griglia ed aggiungere i campi di immissione sistema, file
   // e libreria.
   GridLayoutFormPanel panel = new GridLayoutFormPanel(2);

   LabelFormElement sysPrompt = new LabelFormElement("Server: ");
   TextFormInput system = new TextFormInput("System");
   system.setSize(10);

   LabelFormElement filePrompt = new LabelFormElement("File name: ");
   TextFormInput file = new TextFormInput("File");
   file.setSize(10);

   LabelFormElement libPrompt = new LabelFormElement("Library name: ");
   TextFormInput library = new TextFormInput("Library");
   library.setSize(10);

   panel.addElement(sysPrompt);
   panel.addElement(system);
   panel.addElement(filePrompt);
   panel.addElement(file);
   panel.addElement(libPrompt);
   panel.addElement(library);

   // Aggiungere il pannello al modulo.
   mainForm.addElement(panel);

   // Creare il pulsante di inoltro ed aggiungerlo al modulo.
   mainForm.addElement(new SubmitFormInput("getRecords", "Show Records"));
}
catch (Exception e)
{
   e.printStackTrace ();
}
```

```
      page.append(mainForm.toString());
      page.append("</body></html>");

      return page.toString();
   }

}
```

Il modulo HTML creato dagli esempi precedenti assume questa forma:

```
<table border="0">
<tr>
<td><b>Record number: 1</b></td>
</tr>
<tr>
<td><table border="0">
<tr>
<td><table border="3" cellpadding="2" cellspacing="4">
<tr>
<th>CUSNUM</th>
<td>839283</td>
</tr>
<tr>
<th>LSTNAM</th>
<td>Jones   </td>
</tr>
<tr>
<th>INIT</th>
<td>B D</td>
</tr>
<tr>
<th>STREET</th>
<td>21B NW 135 St</td>
</tr>
<tr>
<th>CITY</th>
<td>Clay  </td>
</tr>
<tr>
<th>STATE</th>
<td>NY</td>
</tr>
<tr>
<th>ZIPCOD</th>
<td>13041</td>
</tr>
<tr>
<th>CDTLMT</th>
<td>400</td>
</tr>
<tr>
<th>CHGCOD</th>
<td>1</td>
</tr>
<tr>
<th>BALDUE</th>
<td>100.00</td>
</tr>
<tr>
<th>CDTDUE</th>
<td>0.00</td>
</tr>
</table>
</td>
<td><font color="#0000ff"> <b><!-- Emissione dalla classe HTMLFormConverter-->
</b></font></td>
</tr>
```

```
</table>
</td>
</tr>
<tr>
<form>
<td><input type="submit" name="getFirstRecord" value="First" />
<input type="submit" name="getPreviousRecord" value="Previous" />
<input type="submit" name="getNextRecord" value="Next" />
<input type="submit" name="getLastRecord" value="Last" />
<br />
</td>
</tr>
<tr>
<td><input type="submit" name="returnToMain" value="Return to Main" />
<br />
</td>
</tr>
</table>
</form>
```

Esempio LightsOn per le classi HTML e servlet

Questo esempio mostra come funzionano le classi HTML e servlet. E' una panoramica generale. Per visualizzare questo esempio, è necessario compilarlo ed eseguirlo con un server web ed un browser attivi.

```
import java.io.IOException;
import java.io.CharArrayWriter;
import java.io.PrintWriter;
import java.sql.*;
import java.util.Enumeration;
import java.util.Hashtable;
import javax.servlet.*;
import javax.servlet.http.*;

import com.ibm.as400.util.html.*;
import com.ibm.as400.util.servlet.*;
import com.ibm.as400.access.*;

/*
Segue un esempio di utilizzo delle classi IBM Toolbox per Java in un servlet.

Schemi di database SQL sul server:

File . . . . . .   LICENSES
Library  . . .     LIGHTSON

  Field            Type          Length   Nulls
  LICENSE          CHARACTER         10   NOT NULL
  USER_ID          CHARACTER         10   NOT NULL WITH DEFAULT
  E_MAIL           CHARACTER         20   NOT NULL
  WHEN_ADDED       DATE                   NOT NULL WITH DEFAULT
  TIME_STAMP       TIMESTAMP              NOT NULL WITH DEFAULT


File . . . . . .   REPORTS
Library  . . .     LIGHTSON

  Field            Type          Length   Nulls
  LICENSE          CHARACTER         10   NOT NULL
  REPORTER         CHARACTER         10   NOT NULL WITH DEFAULT
  DATE_ADDED       DATE                   NOT NULL WITH DEFAULT
  TIME_ADDED       TIME                   NOT NULL WITH DEFAULT
  TIME_STAMP       TIMESTAMP              NOT NULL WITH DEFAULT
  LOCATION         CHARACTER         10   NOT NULL
  COLOR            CHARACTER         10   NOT NULL
  CATEGORY         CHARACTER         10   NOT NULL
*/
```

```
public class LightsOn extends javax.servlet.http.HttpServlet

{
      private AS400 system_;
    private String password_;  // parola d'ordine per il server e per il database SQL
    private java.sql.Connection databaseConnection_;



    public void destroy (ServletConfig config)
  {
           try {
           if (databaseConnection_ != null) {
               databaseConnection_.close();
      }
    }
         catch (Exception e) { e.printStackTrace (); }
  }

    public void doGet (HttpServletRequest request,
                                        HttpServletResponse response)
        throws IOException, ServletException
  {
        HttpSession session = request.getSession();

        response.setContentType("text/html");
        PrintWriter out = response.getWriter();

        out.println(showHtmlMain());

          out.close();
  }

    public void doPost (HttpServletRequest request, HttpServletResponse response)
          throws ServletException, IOException
  {
        HttpSession session = request.getSession(true);
        ServletOutputStream out = response.getOutputStream();
        response.setContentType("text/html");

        Hashtable parameters = getRequestParameters (request);

        if (parameters.containsKey("askingToReport"))
            out.println (showHtmlForReporting ());
        else if (parameters.containsKey("askingToRegister"))
            out.println (showHtmlForRegistering ());
        else if (parameters.containsKey("askingToUnregister"))
            out.println(showHtmlForUnregistering());
        else if (parameters.containsKey("askingToListRegistered"))
            out.println (showHtmlForListingAllRegistered ());
        else if (parameters.containsKey("askingToListReported"))
            out.println (showHtmlForListingAllReported ());
        else if (parameters.containsKey("returningToMain"))
            out.println (showHtmlMain ());


        else { // Nessuno dei precedenti, quindi presupporre che l'utente abbia compilato un modulo
           // e che stia inoltrando le informazioni. Raccogliere le informazioni in entrata ed
                      // eseguire l'azione richiesta.

            if (parameters.containsKey("submittingReport")) {
                String acknowledgement = reportLightsOn (parameters, out);
                out.println (showAcknowledgement(acknowledgement));
      }

            else if (parameters.containsKey("submittingRegistration")) {
```

```
            String acknowledgement = registerLicense (parameters, out);
            out.println (showAcknowledgement(acknowledgement));
    }

          else if (parameters.containsKey("submittingUnregistration")) {
            String acknowledgement = unregisterLicense (parameters, out);
            out.println (showAcknowledgement(acknowledgement));
    }

          else {
            out.println (showAcknowledgement("Error (internal): " +
                                             "Neither Report, Register, " +
                        "Unregister, ListRegistered, or ListReported."));
    }
  }

      out.close(); // Chiudere il flusso di emissione.

}

  //Richiama i parametri da una richiesta servlet HTTP e li comprime in un
  // hashtable per convenienza.
  private static Hashtable getRequestParameters (HttpServletRequest request)
{
      Hashtable parameters = new Hashtable ();
      Enumeration enum = request.getParameterNames();
      while (enum.hasMoreElements()) {
          String key = (String) enum.nextElement();
          String value = request.getParameter (key);
          parameters.put (key, value);
  }
      return parameters;
}


  // Elimina gli spazi vuoti e i trattini da una Stringa e la imposta su caratteri maiuscoli.
  private static String normalize (String oldString)
{
      if (oldString == null || oldString.length() == 0) return null;
      StringBuffer newString = new StringBuffer ();
      for (int i=0; i<oldString.length(); i++) {
          if (oldString.charAt(i) != ' ' && oldString.charAt(i) != '-')
              newString.append (oldString.charAt(i));
  }
      return newString.toString().toUpperCase();
}

  // Compone un elenco di stringhe tra singoli apici.
  private static String quoteList (String[] inList)
{
      StringBuffer outList = new StringBuffer();
      for (int i=0; i<inList.length; i++)
  {
          outList.append ("'" + inList[i] + "'");
          if (i<inList.length-1)
              outList.append (",");
  }
      return outList.toString();
}


  public String getServletInfo ()
{
      return "Lights-On Servlet";
}

  private AS400 getSystem ()
```

```
{
                              try
  {
                        if (system_ == null)
    {
                      system_ = new AS400();

              // Nota: sarebbe meglio richiamare questi valori
              // dal file proprietà.
              String sysName  = "MYSYSTEM";    // TBD
              String userId   = "MYUSERID";    // TBD
              String password = "MYPASSWD";    // TBD

              system_.setSystemName(sysName);
              system_.setUserId(userId);
              system_.setPassword(password);
              password_ = password;

              system_.connectService(AS400.DATABASE);
              system_.connectService(AS400.FILE);
              system_.addPasswordCacheEntry(sysName, userId, password_);
    }
  }
      catch (Exception e) { e.printStackTrace (); system_ = null; }

      return system_;
}


  public void init (ServletConfig config)
{
      boolean rc;

          try {
          super.init(config);

          // Registrare l'unità di controllo JDBC.
            try {
              java.sql.DriverManager.registerDriver(new com.ibm.as400.access.AS400JDBCDriver());
    }
                                catch (Exception e)
    {
              System.out.println("JDBC Driver not found");
    }


  }
                      catch (Exception e) { e.printStackTrace(); }
}

  private void getDatabaseConnection ()
{
      if (databaseConnection_ == null) {
            try {
              databaseConnection_ = java.sql.DriverManager.getConnection(
                                    "jdbc:as400://" + getSystem().getSystemName() + "/" +
                                    "LIGHTSON", getSystem().getUserId(), password_ );

    }
          catch (Exception e) { e.printStackTrace (); }
  }
}


  private String registerLicense (Hashtable parameters, ServletOutputStream out)
{
      String licenseNum = normalize ((String)parameters.get("licenseNum"));
```

```
      String eMailAddress = (String)parameters.get("eMailAddress");
      StringBuffer acknowledgement = new StringBuffer();

      if (licenseNum == null || licenseNum.length() == 0)
          acknowledgement.append ("Error: License number not specified.\n");

      if (eMailAddress == null || eMailAddress.length() == 0)
          acknowledgement.append ("Error: Notification e-mail address not specified.\n");

      if (acknowledgement.length() == 0)
  {
                                try
    {
              // Inserire il nuovo numero di licenza e l'indirizzo e-mail nel database.
            getDatabaseConnection ();
            Statement sqlStatement = databaseConnection_.createStatement();

              // Immettere la richiesta.
      String cmd = "INSERT INTO LICENSES (LICENSE, E_MAIL) VALUES (" +
        quoteList(new String[] {licenseNum, eMailAddress}) + ")";
              sqlStatement.executeUpdate(cmd);
            sqlStatement.close();

              // Conoscere la richiesta.
      acknowledgement.append ("License number " + licenseNum + " has been registered.");
              acknowledgement.append ("Notification e-mail address is: " + eMailAddress);
    }
          catch (Exception e) { e.printStackTrace (); }
  }
      return acknowledgement.toString();
}


  private String unregisterLicense (Hashtable parameters, ServletOutputStream out)
{
      String licenseNum = normalize ((String)parameters.get("licenseNum"));
      StringBuffer acknowledgement = new StringBuffer();

      if (licenseNum == null || licenseNum.length() == 0)
          acknowledgement.append ("Error: License number not specified.\n");

      if (acknowledgement.length() == 0)
  {
                                try
    {
              // Eliminare il numero di licenza e l'indirizzo e-mail specificati dal database.
            getDatabaseConnection ();
            Statement sqlStatement = databaseConnection_.createStatement();

              // Cancellare la(e) riga(he) dal database LICENSES.
              String cmd = "DELETE FROM LICENSES WHERE LICENSE = '" + licenseNum + "'";
              sqlStatement.executeUpdate(cmd);
            sqlStatement.close();

              // Conoscere la richiesta.
      acknowledgement.append ("License number " + licenseNum + " has been unregistered.");
    }
          catch (Exception e) { e.printStackTrace (); }
  }
      return acknowledgement.toString();
}


  private String reportLightsOn (Hashtable parameters, ServletOutputStream out)
{
      String licenseNum = normalize ((String)parameters.get("licenseNum"));
      String location = (String)parameters.get("location");
```

```
      String color = (String)parameters.get("color");
      String category = (String)parameters.get("category");
      StringBuffer acknowledgement = new StringBuffer();
      if (licenseNum == null || licenseNum.length() == 0)
    acknowledgement.append ("Error: License number not specified.");

      if (acknowledgement.length() == 0)
  {
                               try
    {
              // Riportare "lights on" per il veicolo specificato.
            getDatabaseConnection ();
            Statement sqlStatement = databaseConnection_.createStatement();

              // Aggiungere una voce al database REPORTS.
      String cmd = "INSERT INTO REPORTS (LICENSE, LOCATION, COLOR, CATEGORY) VALUES (" +
        quoteList(new String[] {licenseNum, location, color, category}) + ")";
              sqlStatement.executeUpdate(cmd);
            sqlStatement.close();

              // Conoscere la richiesta.
      acknowledgement.append ("License number " + licenseNum + " has been reported.  Thanks!");
    }
          catch (Exception e) { e.printStackTrace (); }
  }
      return acknowledgement.toString();
}


  private String showHeader (String title)
{
      StringBuffer page = new StringBuffer();
  page.append("<html><head><title>" + title + "</title>");
  page.append("</head><body bgcolor=\"blanchedalmond\">");
      return page.toString ();
}


  private String showAcknowledgement (String acknowledgement)
{
      String title = "Acknowledgement";
      StringBuffer page = new StringBuffer();
      page.append (showHeader (title));

          try {
          HTMLForm form = new HTMLForm("LightsOn");
        GridLayoutFormPanel grid = new GridLayoutFormPanel();
          HTMLText text = new HTMLText(acknowledgement);
    if (acknowledgement.startsWith("Error"))
               text.setBold(true);
          grid.addElement(text);
          grid.addElement(new SubmitFormInput("returningToMain", "Home"));
          form.addElement(grid);
          page.append(form.toString());
  }
        catch (Exception e) { e.printStackTrace (); }
  page.append("</body></html>");
      return page.toString();
}


  private String showHtmlMain ()
{
      String title = "Lights-On tool";
      StringBuffer page = new StringBuffer();
      page.append (showHeader (title));
```

```
  page.append("<h1>" + title + "</h1>");


        // Creare l'oggetto HTML Form.
        HTMLForm mainForm = new HTMLForm("LightsOn");
      GridLayoutFormPanel grid = new GridLayoutFormPanel();

          try {
          // Impostare in modo che doPost() venga chiamato quando si inoltra il modulo.
          mainForm.setMethod(HTMLForm.METHOD_POST);

          // Creare alcuni pulsanti.
    grid.addElement(new SubmitFormInput("askingToReport", "Report a vehicle with lights on"));
    grid.addElement(new SubmitFormInput("askingToRegister", "Register my license number"));
    grid.addElement(new SubmitFormInput("askingToUnregister", "Unregister my license number"));
    grid.addElement(new SubmitFormInput("askingToListRegistered", "List all registered licenses"));
    grid.addElement(new SubmitFormInput("askingToListReported", "List all vehicles with lights on"));

          mainForm.addElement(grid);
  }
        catch (Exception e) { e.printStackTrace (); }

      page.append(mainForm.toString());
  page.append("</body></html>");

      return page.toString();
}


  private String showHtmlForReporting ()
{
      String title = "Report a vehicle with lights on";
      StringBuffer page = new StringBuffer();
      page.append (showHeader (title));

  page.append("<h1>" + title + "</h1>");

        // Creare l'oggetto HTML Form.
      HTMLForm reportForm = new HTMLForm("LightsOn");
      GridLayoutFormPanel grid = new GridLayoutFormPanel(2);

          try {
          // Impostare in modo che doPost() venga chiamato quando si inoltra il modulo.
          reportForm.setMethod(HTMLForm.METHOD_POST);

          TextFormInput licenseNum = new TextFormInput("licenseNum");
          licenseNum.setSize(10);
          licenseNum.setMaxLength(10);

             // Aggiungere elementi al formato della riga
          grid.addElement(new LabelFormElement("Vehicle license number:"));
          grid.addElement(licenseNum);

          // Creare un gruppo di pallini e aggiungerli.
          RadioFormInputGroup colorGroup = new RadioFormInputGroup("color");

          colorGroup.add("color", "white", "white", true);
          colorGroup.add("color", "black", "black", false);
          colorGroup.add("color", "gray", "gray", false);
          colorGroup.add("color", "red", "red", false);
          colorGroup.add("color", "yellow", "yellow", false);
          colorGroup.add("color", "green", "green", false);
          colorGroup.add("color", "blue", "blue", false);
          colorGroup.add("color", "brown", "brown", false);

          // Creare un elenco di selezione per la categoria del veicolo.
          SelectFormElement category = new SelectFormElement("category");
```

```
          category.addOption("sedan", "sedan", true);
          category.addOption("convertible", "convertibl"); // campo di 10 car in DB
          category.addOption("truck", "truck");
          category.addOption("van", "van");
          category.addOption("SUV", "SUV");
          category.addOption("motorcycle", "motorcycle");
          category.addOption("other", "other");

          // Creare un elenco di selezione per l'ubicazione del veicolo (numero di edificio).
          SelectFormElement location = new SelectFormElement("location");
          location.addOption("001", "001", true);
          location.addOption("002", "002");
          location.addOption("003", "003");
          location.addOption("005", "005");
          location.addOption("006", "006");
          location.addOption("015", "015");

          grid.addElement(new LabelFormElement("Color:"));
          grid.addElement(colorGroup);

          grid.addElement(new LabelFormElement("Vehicle type:"));
          grid.addElement(category);

          grid.addElement(new LabelFormElement("Building:"));
          grid.addElement(location);

          grid.addElement(new SubmitFormInput("submittingReport", "Submit report"));
          grid.addElement(new SubmitFormInput("returningToMain", "Home"));

          reportForm.addElement(grid);
  }
        catch (Exception e) { e.printStackTrace (); }

      page.append(reportForm.toString());

  page.append("</body></html>");

      return page.toString();
}


  private String showHtmlForRegistering ()
{
      String title = "Register my license number";
      StringBuffer page = new StringBuffer();
      page.append (showHeader (title));

  page.append("<h1>" + title + "</h1>");

        // Creare l'oggetto HTML Form.
      HTMLForm registrationForm = new HTMLForm("LightsOn");

      // Impostare il layout di un pannello a due colonne, nel quale disporre
        // gli elementi HTML generati.
      GridLayoutFormPanel grid = new GridLayoutFormPanel(2);

          try {
          // Impostare in modo che doPost() venga chiamato quando si inoltra il modulo.
          registrationForm.setMethod(HTMLForm.METHOD_POST);

          TextFormInput licenseNum = new TextFormInput("licenseNum");
          licenseNum.setSize(10);
          licenseNum.setMaxLength(10);

          TextFormInput eMailAddress = new TextFormInput("eMailAddress");
          eMailAddress.setMaxLength(20);
```

```
          grid.addElement(new LabelFormElement("License number:"));
          grid.addElement(licenseNum);

          grid.addElement(new LabelFormElement("E-mail notification address:"));
          grid.addElement(eMailAddress);

          grid.addElement(new SubmitFormInput("submittingRegistration", "Register"));
          grid.addElement(new SubmitFormInput("returningToMain", "Home"));

          registrationForm.addElement(grid);
  }
        catch (Exception e) { e.printStackTrace (); }

      page.append(registrationForm.toString());

  page.append("</body></html>");

      return page.toString();
}


  private String showHtmlForUnregistering ()
{
      String title = "Unregister my license number";
      StringBuffer page = new StringBuffer();
      page.append (showHeader (title));

  page.append("<h1>" + title + "</h1>");

        // Creare l'oggetto HTML Form.
      HTMLForm unregistrationForm = new HTMLForm("LightsOn");
      GridLayoutFormPanel grid = new GridLayoutFormPanel(2);

          try {
          // Impostare in modo che doPost() venga chiamato quando si inoltra il modulo.
          unregistrationForm.setMethod(HTMLForm.METHOD_POST);

          // Creare l'oggetto LineLayoutFormPanel.
          TextFormInput licenseNum = new TextFormInput("licenseNum");
          licenseNum.setSize(10);
          licenseNum.setMaxLength(10);

          grid.addElement(new LabelFormElement("Vehicle license number:"));
          grid.addElement(licenseNum);

          grid.addElement(new SubmitFormInput("submittingUnregistration", "Unregister"));
          grid.addElement(new SubmitFormInput("returningToMain", "Home"));

          unregistrationForm.addElement(grid);
  }
          catch (Exception e) {
            e.printStackTrace();
          CharArrayWriter cWriter = new CharArrayWriter();
          PrintWriter pWriter = new PrintWriter (cWriter, true);
          e.printStackTrace (pWriter);
          page.append (cWriter.toString());
  }

      page.append(unregistrationForm.toString());

  page.append("</body></html>");

      return page.toString();
}


  private String showHtmlForListingAllRegistered ()
```

```
{
      String title = "All registered licenses";
      StringBuffer page = new StringBuffer();
      page.append (showHeader (title));

                              try
  {
         // Creare l'oggetto HTML Form.
         HTMLForm mainForm = new HTMLForm("LightsOn");

         // Impostare il layout di un pannello a colonna singola, nel quale disporre
         // gli elementi HTML generati.
        GridLayoutFormPanel grid = new GridLayoutFormPanel();

         // Specificare il layout per la tabella generata.
            HTMLTable table = new HTMLTable();
         table.setAlignment(HTMLConstants.LEFT);
         table.setBorderWidth(3);

         // Creare e aggiungere il titolo e l'intestazione della tabella.
         HTMLTableCaption caption = new HTMLTableCaption();
         caption.setAlignment(HTMLConstants.TOP);
         caption.setElement(title);
         table.setCaption(caption);
         table.setHeader(new String[] { "License", "Date added" } );

         // Creare il programma di conversione, che creerà la tabella HTML dalla
         // serie di risultati restituita dall'interrogazione database.
       HTMLTableConverter converter = new HTMLTableConverter();
       converter.setTable(table);

         getDatabaseConnection ();
         Statement sqlStatement = databaseConnection_.createStatement();

         // Eseguire innanzitutto una preinterrogazione al database per verificare che nono sia vuoto.
         String query = "SELECT COUNT(*) FROM LICENSES";
         ResultSet rs = sqlStatement.executeQuery (query);
         rs.next();  // cursore posizionato sulla prima riga
         int rowCount = rs.getInt(1);

         if (rowCount == 0) {
      page.append ("<font size=4 color=red>No vehicles have been reported.</font>");
    }
         else {
             query = "SELECT LICENSE,WHEN_ADDED FROM LICENSES";
             rs = sqlStatement.executeQuery (query);
             SQLResultSetRowData rowData = new SQLResultSetRowData (rs);
             HTMLTable[] generatedHtml = converter.convertToTables(rowData);
             grid.addElement(generatedHtml[0]);
    }
         sqlStatement.close();
         // Nota: non chiudere l'istruzione prima di aver finito di utilizzare la serie di risultati.

         grid.addElement(new SubmitFormInput("returningToMain", "Home"));

         mainForm.addElement(grid);
        page.append(mainForm.toString());
  }
        catch (Exception e) { e.printStackTrace (); }
  page.append("</body></html>");
      return page.toString();
}


  private String showHtmlForListingAllReported ()
{
      String title = "All vehicles with lights on";
```

```
        StringBuffer page = new StringBuffer();
        page.append (showHeader (title));

                                try
    {
            // Creare l'oggetto HTML Form.
            HTMLForm form = new HTMLForm("LightsOn");

            // Impostare il layout di un pannello a colonna singola, nel quale disporre
            // gli elementi HTML generati.
          GridLayoutFormPanel grid = new GridLayoutFormPanel();

            // Specificare il layout per la tabella generata.
               HTMLTable table = new HTMLTable();
            table.setAlignment(HTMLConstants.LEFT);
            table.setBorderWidth(3);

            // Creare e aggiungere il titolo e l'intestazione della tabella.
            HTMLTableCaption caption = new HTMLTableCaption();
            caption.setAlignment(HTMLConstants.TOP);
            caption.setElement(title);
            table.setCaption(caption);
            table.setHeader(new String[] { "License", "Color", "Category", "Date", "Time" } );

            // Creare il programma di conversione, che genera la tabella HTML dalla
            // serie di risultati restituita dall'interrogazione database.
         HTMLTableConverter converter = new HTMLTableConverter();
         converter.setTable(table);

            getDatabaseConnection ();
            Statement sqlStatement = databaseConnection_.createStatement();

            // Eseguire innanzitutto una preinterrogazione al database per verificare che nono sia vuoto.
            String query = "SELECT COUNT(*) FROM REPORTS";
            ResultSet rs = sqlStatement.executeQuery (query);
            rs.next();  // cursore posizionato sulla prima riga
            int rowCount = rs.getInt(1);

            if (rowCount == 0) {
          page.append ("<font size=4 color=red>No vehicles have been reported.</font>");
       }
            else {
                query = "SELECT LICENSE,COLOR,CATEGORY,DATE_ADDED,TIME_ADDED FROM REPORTS";
                rs = sqlStatement.executeQuery (query);
                SQLResultSetRowData rowData = new SQLResultSetRowData (rs);
                HTMLTable[] generatedHtml = converter.convertToTables(rowData);
                grid.addElement(generatedHtml[0]);
       }
            sqlStatement.close();
            // Nota: non chiudere l'istruzione prima di aver finito di utilizzare la serie di risultati.

            grid.addElement(new SubmitFormInput("returningToMain", "Home"));
            form.addElement(grid);
            page.append(form.toString());
    }
          catch (Exception e) { e.printStackTrace (); }
      page.append("</body></html>");
          return page.toString();
   }

}
```

Esempi semplici di programmazione

Questi esempi mostrano alcuni modi per codificare i propri programmi Java utilizzando le classi IBM Toolbox per Java. Rivolti a programmatori che stanno appena iniziando ad utilizzare le classi IBM Toolbox per Java, questi esempi includono spiegazioni dettagliate sulle righe chiave nel codice.

Se si desidera avere un aiuto introduttivo, consultare Come scrivere il primo programma IBM Toolbox per Java.

Per collegamenti a molti altri esempi forniti nelle informazioni IBM Toolbox per Java, consultare Esempi di codice.

Utilizzare l'elenco seguente per visualizzare gli esempi semplici di programmazione:

- Chiamata comandi
- Utilizzo delle code messaggi
- Utilizzo dell'accesso a livello record
- Utilizzo delle classi JDBC per creare e riempire una tabella
- Visualizzazione di un elenco di lavori server in una GUI

Il seguente esonero di responsabilità si applica a tutti gli esempi IBM Toolbox per Java:

Esonero di responsabilità per gli esempi di codice

L'IBM fornisce una licenza non esclusiva per utilizzare tutti gli esempi del codice di programmazione da cui creare funzioni simili personalizzate, in base a richieste specifiche.

Questo codice di esempio è fornito dall'IBM con la sola funzione illustrativa. Questi esempi non sono stati interamente testati in tutte le condizioni. IBM, perciò, non fornisce nessun tipo di garanzia o affidabilità implicita, rispetto alla funzionalità o alle funzioni di questi programmi.

Tutti i programmi qui contenuti vengono forniti all'utente ″COSI' COME SONO″ senza garanzie di alcun tipo. Le garanzie implicite di non contraffazione, commerciabilità e adeguatezza a scopi specifici sono espressamente vietate.

Come scrivere il primo programma IBM Toolbox per Java

Per iniziare questo semplice esercizio, è necessario installare Java sulla stazione di lavoro. E' possibile stabilire quale versione si desidera installare controllando Requisiti per l'esecuzione di applicazioni Java.

Dopo aver installato Java sul client, completare le seguenti attività:

1. Copiare jt400.jar sulla stazione di lavoro.
2. Aggiungere jt400.jar a CLASSPATH sulla stazione di lavoro aggiungendo il percorso completo del file JAR a CLASSPATH. Ad esempio, quando il file jt400.jar risiede nell'indirizzario c:\lib sulla stazione di lavoro (che esegue Windows), aggiungere quanto segue alla fine dell'istruzione CLASSPATH:

   ```
   ;c:\lib\jt400.jar
   ```

3. Aprire un editor di testo ed inserire il primo esempio di programmazione semplice

   Nota: accertarsi di non includere il testo che si riferisce alle Note (ad esempio, Nota 1, Nota 2 e così via). Salvare il nuovo documento con il nome CmdCall.java.
4. Avviare una sessione comandi sulla stazione di lavoro ed utilizzare il comando riportato di seguito per compilare l'esempio di programmazione semplice:

   ```
   javac CmdCall.java
   ```

5. Nella sessione comandi, immettere il comando riportato di seguito per eseguire l'esempio di programmazione semplice:

```
java CmdCall
```

[Esempi semplici di programmazione]

Esempio: utilizzo di CommandCall

Utilizzare quanto segue come esempio per il proprio programma.

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per informazioni legali importanti.

```
////////////////////////////////////////////////////////////////////////////
//
// Esempio di utilizzo della classe Access di IBM Toolbox per Java, CommandCall.
//
// Questo sorgente è un esempio di "Job List" di IBM Toolbox per Java.
//
////////////////////////////////////////////////////////////////////////////
//
// Le classi access di IBM Toolbox per Java si trovano in
// com.ibm.as400.access.package. Importare questo pacchetto per utilizzare le classi IBM Toolbox per
// Java.
//
////////////////////////////////////////////////////////////////////////////

import com.ibm.as400.access.*;

public class CmdCall
{
       public static void main (String[] args)
   {
      // Come altre classi Java, le classi IBM Toolbox per Java emettono
      // eccezioni in caso di errori. Tali eccezioni
      // devono essere individuate dai programmi che utilizzano IBM Toolbox per Java.
      try Note 1
      {
                  AS400 system = new AS400();

         CommandCall cc = new CommandCall(system); Note 2

         cc.run("CRTLIB MYLIB"); Note 3

         AS400Message[] ml = cc.getMessageList(); Note 4

                  for (int i=0; i<ml.length; i++)
         {
            System.out.println(ml[i].getText()); Note 5
         }
      }
                                  catch (Exception e)
      {
                 e.printStackTrace();
      }

               System.exit(0);
   }
}
```

1. IBM Toolbox per Java utilizza l'oggetto ″AS400″ per identificare il server di destinazione. Se si crea l'oggetto AS400 senza parametri, IBM Toolbox per Java richiede il nome del sistema, l'ID utente e la parola d'ordine. La classe AS400 inoltre include un programma di creazione che prende il nome di sistema, l'ID utente e la parola d'ordine.

2. Utilizzare l'oggetto CommandCall di IBM Toolbox per Java per inviare i comandi al server. Quando si crea l'oggetto CommandCall, si inoltra un oggetto AS400 in modo tale che sappia quale server sia la destinazione del comando.
3. Utilizzare il comando run() sull'oggetto CommandCall per eseguire il comando.
4. Il risultato dell'esecuzione di un comando è un elenco di messaggi OS/400. IBM Toolbox per Java rappresenta tali messaggi come oggetti AS400Message. Quando il comando è completato, si ricevono i messaggi risultanti dall'oggetto CommandCall.
5. Stampare il testo messaggio. E' inoltre disponibile l'ID messaggio, la severità messaggio e altre informazioni. Questo programma stampa solo il testo messaggio.

Esempio: utilizzo delle code messaggi (parte 1 di 3)

[Parte successiva]

Utilizzare quanto segue come esempio per il proprio programma.

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
////////////////////////////////////////////////////////////////////////////
//
// Esempio di utilizzo della funzione Message Queue dell'IBM Toolbox per Java
//
// Questo sorgente è un esempio di "Message Queue" di IBM Toolbox per Java.
//
////////////////////////////////////////////////////////////////////////////

package examples;  Note 1


import java.io.*;
import java.util.*;

import com.ibm.as400.access.*;  Note 2

public class displayMessages extends Object
{

   public static void main(String[] parameters)  Note 3
   {
            displayMessages me = new displayMessages();
      me.Main(parameters);  Note 4

      System.exit(0);  Note 5
   }


      void displayMessage()
   {
   }


      void Main(String[] parms)
   {
      try  Note 6
      {

                              // Il codice IBM Toolbox per Java viene inserito qui
      }
                                  catch (Exception e)
      {
         e.printStackTrace();  Note 7
      }
   }
}
```

1. Questa classe si trova nel pacchetto 'esempi'. Java utilizza i pacchetti per evitare dei conflitti con i nomi tra i file di classe Java.
2. Questa riga rende tutte le classi IBM Toolbox per Java nel pacchetto Access disponibili per questo programma. Le classi nel pacchetto Access hanno il prefisso comune **com.ibm.as400**.Mediante un'istruzione Import, il programma può fare riferimento ad una classe utilizzandone il nome, non il nome completo. Ad esempio, è possibile fare riferimento alla classe AS400 utilizzando AS400, invece di com.ibm.as400.AS400.
3. Questa classe possiede un metodo **main**; perciò, può essere eseguita come un'applicazione.Per richiamare il programma, è necessario eseguire **java examples.displayMessages**.Notare che i caratteri maiuscoli/minuscoli devono corrispondere quando viene eseguito il programma. Poiché viene utilizzata una classe IBM Toolbox per Java, jt400.zip deve trovarsi nella variabile di ambiente classpath.
4. Il metodo main menzionato nella Nota 3 è statico. Una delle restrizioni dei metodi statici è data dal fatto che i metodi statici possono richiamare solo altri metodi statici nella propria classe. Per evitare questa restrizione, molti programmi java creano un oggetto e successivamente effettuano un processo di inizializzazione nel metodo denominato **Main**. Il metodo Main() può richiamare qualsiasi altro metodo nell'oggetto displayMessages.
5. IBM Toolbox per Java crea sottoprocessi per conto dell'applicazione per espletare l'attività di IBM Toolbox per Java. Se il programma non emette **System.exit(0)** al momento della chiusura, il programma potrebbe non terminare correttamente.Ad esempio, supponiamo che questo programma sia stata eseguito da una richiesta di dos Windows 95. Senza questa riga, la richiesta comandi non verrà restituita alla fine del programma. L'utente deve immettere Ctrl-C per richiamare una richiesta comandi.
6. Il codice IBM Toolbox per Java emette delle eccezioni che il programma deve rilevare.
7. Questo programma visualizza il testo dell'eccezione mentre il programma sta eseguendo l'elaborazione dell'errore. Le eccezioni emesse dall'IBM Toolbox per Java vengono convertite, quindi il testo dell'eccezione sarà nella stessa lingua della stazione di lavoro.

[Parte successiva]

Esempio: utilizzo delle code messaggi (parte 2 di 3)

[Parte precedente | Parte successiva]

Utilizzare quanto segue come esempio per il proprio programma.

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
////////////////////////////////////////////////////////////////////////////
//
// Esempio di utilizzo della funzione Message Queue dell'IBM Toolbox per Java
//
// Questo sorgente è un esempio di "Message Queue" di IBM Toolbox per Java.
//
////////////////////////////////////////////////////////////////////////////

package examples;


import java.io.*;
import java.util.*;
import com.ibm.as400.access.*;

public class displayMessages extends Object
{

      public static void main(String[] parameters)
   {
            displayMessages me = new displayMessages();
```

```
            me.Main(parameters);

                  System.exit(0);
   }


      void displayMessage()
   {
   }


      void Main(String[] parms)
   {
                                 try
      {

          AS400 system = new AS400();  Note 1


                     if (parms.length > 0)
                        system.setSystemName(parms[0]);  Note 2

      }
                                 catch (Exception e)
      {
                e.printStackTrace();
      }
   }
}
```

1. Un programma utilizza l'oggetto **AS400** per designare il server al quale collegarsi.Con una eccezione, tutti i programmi che necessitano di risorse da un server devono avere un oggetto AS400. L'eccezione è JDBC. Se il programma utilizza JDBC, l'unità di controllo JDBC IBM Toolbox per Java crea l'oggetto AS400 per il programma.
2. Questo programma presuppone che il primo parametro della riga comandi corrisponda al nome del server. Se un parametro viene passato al programma, il metodo **setSystemName** dell'oggetto AS400 viene utilizzato per impostare il nome di sistema. L'oggetto AS400 necessita anche delle informazioni di collegamento al server:
   - Se il programma è in esecuzione su una stazione di lavoro, il programma IBM Toolbox per Java richiede all'utente un ID utente e una parola d'ordine. **Nota:** se un nome sistema non viene specificato come parametro della riga comandi, l'oggetto AS400 richiede anche il nome sistema.
   - Se il programma è in esecuzione sulla JVM iSeries, viene utilizzato l'ID utente e la parola d'ordine dell'utente che sta eseguendo il programma Java. In tal caso, l'utente non specifica un nome sistema, ma assegna al nome di sistema, come valore predefinito, il nome del sistema su cui il programma è in esecuzione.

[Parte precedente | Parte successiva]

Esempio: utilizzo delle code messaggi (parte 3 di 3)

[Parte precedente]

Utilizzare quanto segue come esempio per il proprio programma.

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
//////////////////////////////////////////////////////////////////////////////
//
// Esempio di utilizzo della funzione Message Queue dell'IBM Toolbox per Java
//
// Questo sorgente è un esempio di "Message Queue" di IBM Toolbox per Java.
//
```

```
////////////////////////////////////////////////////////////////////////////

package examples;


import java.io.*;
import java.util.*;
import com.ibm.as400.access.*;

public class displayMessages extends Object
{

      public static void main(String[] parameters)
   {
            displayMessages me = new displayMessages();

            me.Main(parameters);

                  System.exit(0);
   }


      void displayMessage()
   {
   }


      void Main(String[] parms)
   {
                                 try
      {
                  AS400 system = new AS400();

         if (parms.length > 0)
                        system.setSystemName(parms[0]);

         MessageQueue queue = new MessageQueue(system, MessageQueue.CURRENT);  Note 1


                  Enumeration e = queue.getMessages();  Note 2

                        while (e.hasMoreElements())
                  {

                      QueuedMessage message = (QueuedMessage) e.nextElement();  Note 3
                      System.out.println(message.getText());  Note 4
                   }
               }
                                       catch (Exception e)
               {
                         e.printStackTrace();
               }
         }
}
```

1. L'obbiettivo di questo programma è quello di visualizzare i messaggi in una coda messaggi del server. Per questa attività, viene utilizzato l'oggetto **MessageQueue** dell'IBM Toolbox per Java.Quando viene creato l'oggetto della coda messaggi, i parametri sono l'oggetto AS400 e il nome coda messaggi. L'oggetto AS400 indica quale server contiene la risorsa e il nome della coda messaggi identifica la coda messaggi sul server. In questo caso, viene utilizzata una costante, la quale informa l'oggetto coda messaggi di accedere alla coda dell'utente collegato.
2. L'oggetto coda messaggi richiama un elenco di messaggi dal server. Viene effettuato un collegamento al server a questo punto.
3. Rimuovere un messaggio dall'elenco. Il messaggio si trova nell'oggetto QueuedMessage del programma IBM Toolbox per Java.

4. Stampare il testo del messaggio.

[Parte precedente]

Esempio: utilizzo dell'accesso a livello record (parte 1 di 2)

[Parte successiva]

Utilizzare quanto segue come esempio per il proprio programma.

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
////////////////////////////////////////////////////////////////////////////
//
// Esempio di accesso a livello record.
Questo programma richiederà all'utente
// il nome del server ed il file da visualizzare.  Il file deve esistere
// e contenere record.  Ogni record nel file verrà visualizzato
// in System.out.
//
// Sintassi di chiamata: java RLSequentialAccessExample
//
// Questo sorgente è un esempio di "RecordLevelAccess" IBM Toolbox per Java
//
////////////////////////////////////////////////////////////////////////////

import java.io.*;
import java.util.*;
import com.ibm.as400.access.*;

public class RLSequentialAccessExample
{
      public static void main(String[] parameters)
   {
            BufferedReader inputStream = new BufferedReader(new InputStreamReader(System.in),1);

      String systemName = "";
      String library = "";
      String file = "";
      String member = "";

          System.out.println();
                                try
      {
        System.out.print("System name: ");
              systemName = inputStream.readLine();

        System.out.print("Library in which the file exists: ");
              library = inputStream.readLine();

        System.out.print("File name: ");
                file = inputStream.readLine();

        System.out.print("Member name (press enter for first member): ");
                member = inputStream.readLine();
        if (member.equals(""))
        {
          member = "*FIRST";
        }

            System.out.println();
      }
                                catch (Exception e)
      {
        System.out.println("Error obtaining user input.");
                e.printStackTrace();
```

```
            System.exit(0);
}


AS400 system = new AS400(systemName);  Note 1
                           try
{
       system.connectService(AS400.RECORDACCESS);
}
  catch(Exception e)
{
  System.out.println("Unable to connect for record level access.");
  System.out.println("Check the programmer's guide setup file for
    special instructions regarding record level access");
         e.printStackTrace();
            System.exit(0);
}


QSYSObjectPathName filePathName = new QSYSObjectPathName(library, file, member, "MBR");  Note 2


SequentialFile theFile = new SequentialFile(system, filePathName.getPath());  Note 3


AS400FileRecordDescription recordDescription =
     new AS400FileRecordDescription(system, filePathName.getPath());
                           try
{
  RecordFormat[] format = recordDescription.retrieveRecordFormat();  Note 4


  theFile.setRecordFormat(format[0]);  Note 5


  theFile.open(AS400File.READ_ONLY, 100, AS400File.COMMIT_LOCK_LEVEL_NONE);  Note 6

  System.out.println("Displaying file " + library.toUpperCase() + "/" +
    file.toUpperCase() + "(" + theFile.getMemberName().trim() + "):");

  Record record = theFile.readNext();  Note 7
         while (record != null)
  {
            System.out.println(record);
            record = theFile.readNext();
  }
      System.out.println();


  theFile.close();  Note 8


  system.disconnectService(AS400.RECORDACCESS);  Note 9
}
                           catch (Exception e)
{
  System.out.println("Error occurred attempting to display the file.");
         e.printStackTrace();

                             try
  {
           // Chiudere il file
           theFile.close();
  }
       catch(Exception x)
  {
  }
```

```
            system.disconnectService(AS400.RECORDACCESS);
                    System.exit(0);
      }

            // Assicurarsi che l'applicazione termini; consultare il readme per i dettagli
                  System.exit(0);
    }
  }
```

1. Questa riga di codice crea un oggetto AS400 e si collega al servizio di accesso a livello record.
2. Questa riga crea un oggetto QSYSObjectPathName che ottiene il formato nome percorso IFS (integrated file system) dell'oggetto da visualizzare.
3. Questa istruzione crea un oggetto che rappresenta un file sequenziale esistente sul server al quale si è collegati. Questo file sequenziale è il file che verrà visualizzato.
4. Queste righe richiamano il formato record del file.
5. Questa riga imposta il formato record per il file.
6. Questa riga apre per la lettura il file selezionato. Leggerà 100 record alla volta, quando possibile.
7. Questa riga di codice legge ogni record in sequenza.
8. Questa riga chiude il file.
9. Questa riga si scollega dal servizio di accesso al livello record.

[Parte successiva]

Esempio: utilizzo dell'accesso a livello record (parte 2 di 2)

[Parte precedente]

Utilizzare quanto segue come esempio per il proprio programma.

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
////////////////////////////////////////////////////////////////////////////////
//
// Esempio di accesso a livello record.
//
// Sintassi di chiamata: java RLACreateExample
//
////////////////////////////////////////////////////////////////////////////////

import java.io.*;
import java.util.*;
import com.ibm.as400.access.*;

public class RLACreateExample
{
      public static void main (String[] args)
  {
               AS400 system = new AS400 (args[0]);
    String filePathName = "/QSYS.LIB/MYLIB.LIB/MYFILE.FILE/MBR1.MBR";  Note 1

                                try
    {
            SequentialFile theFile = new SequentialFile(system, filePathName);

            // Inizio Nota due
      CharacterFieldDescription lastNameField =
         new CharacterFieldDescription(new AS400Text(20), "LNAME");
      CharacterFieldDescription firstNameField =
```

```
        new CharacterFieldDescription(new AS400Text(20), "FNAME");
      BinaryFieldDescription yearsOld =
        new BinaryFieldDescription(new AS400Bin4(), "AGE");

      RecordFormat fileFormat = new RecordFormat("RF");
            fileFormat.addFieldDescription(lastNameField);
            fileFormat.addFieldDescription(firstNameField);
            fileFormat.addFieldDescription(yearsOld);

      theFile.create(fileFormat, "A file of names and ages");  Note 2
            // Fine Nota due


            theFile.open(AS400File.READ_WRITE, 1, AS400File.COMMIT_LOCK_LEVEL_NONE);

            // Inizio Nota tre
            Record newData = fileFormat.getNewRecord();
      newData.setField("LNAME", "Doe");
      newData.setField("FNAME", "John");
      newData.setField("AGE", new Integer(63));

      theFile.write(newData);  Note 3
            // Fine Nota tre

              theFile.close();
    }
      catch(Exception e)
    {
      System.out.println("An error has occurred: ");
              e.printStackTrace();
    }

        system.disconnectService(AS400.RECORDACCESS);

                System.exit(0);
  }
}
```

1. (args[0]) nella riga precedente e MYFILE.FILE sono parti di codice che rappresentano dei prerequisiti per il resto dell'esempio da eseguire. Il programma presume che la libreria MYLIB esista sul server e che l'utente abbia accesso ad essa.
2. Il testo nei commenti Java etichettato ″Inizio nota due″ e ″Fine nota due″ indica come creare per proprio conto un formato record invece di richiamarlo da un file esistente. L'ultima riga in questo blocco crea il file sul server.
3. Il testo nei commenti Java etichettati ″Inizio nota tre″ e ″Fine nota tre″ indica un modo per creare un record e scriverlo in un file.

[Parte precedente]

Esempio: utilizzo delle classi JDBC per creare e popolamento di una tabella (parte 1 di 2)

[Parte successiva]

Utilizzare quanto segue come esempio per il proprio programma.

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per informazioni legali importanti.

```
////////////////////////////////////////////////////////////////////////////
//
// Esempio JDBCPopulate. Questo programma utilizza l'unità di controllo JDBC IBM Toolbox per Java
// per creare e popolare una tabella.
//
// Sintassi del comando:
```

```
//    JDBCPopulate system collectionName tableName
//
// Ad esempio,
//    JDBCPopulate MySystem MyLibrary MyTable
//
// Questo sorgente è un esempio di unità di controllo JDBC IBM Toolbox per Java.
//
//////////////////////////////////////////////////////////////////////////////

import java.sql.*;

public class JDBCPopulate
{

        private static final String words[]
        = { "One",      "Two",      "Three",    "Four",     "Five",
            "Six",      "Seven",    "Eight",    "Nine",     "Ten",
            "Eleven",   "Twelve",   "Thirteen", "Fourteen", "Fifteen",
            "Sixteen",  "Seventeen","Eighteen", "Nineteen", "Twenty" };

       public static void main (String[] parameters)
    {

                if (parameters.length != 3) {
                     System.out.println("");
                     System.out.println("Usage:");
                     System.out.println("");
            System.out.println("   JDBCPopulate system collectionName tableName");
                     System.out.println("");
                     System.out.println("");
                     System.out.println("For example:");
                     System.out.println("");
                     System.out.println("");
            System.out.println("   JDBCPopulate MySystem MyLibrary MyTable");
                     System.out.println("");
                        return;
        }

                      String system = parameters[0];
                    String collectionName   = parameters[1];
                    String tableName        = parameters[2];

                    Connection connection   = null;

                    try {


            DriverManager.registerDriver(new com.ibm.as400.access.AS400JDBCDriver());  Note 1


            connection = DriverManager.getConnection ("jdbc:as400://"
                + system + "/" + collectionName);  Note 2

                    try {
                                Statement dropTable = connection.createStatement ();
                dropTable.executeUpdate ("DROP TABLE " + tableName);  Note 3
            }
                        catch (SQLException e) {
            }


                        Statement createTable = connection.createStatement ();
            createTable.executeUpdate ("CREATE TABLE " + tableName
                + " (I INTEGER, WORD VARCHAR(20), SQUARE INTEGER, "
                + " SQUAREROOT DOUBLE)");  Note 4
```

```
            PreparedStatement insert = connection.prepareStatement ("INSERT INTO "
                + tableName + " (I, WORD, SQUARE, SQUAREROOT) "
                + " VALUES (?, ?, ?, ?)");  Note 5


                        for (int i = 1; i <= words.length; ++i) {
                                insert.setInt (1, i);
                                insert.setString (2, words[i-1]);
                                insert.setInt (3, i*i);
                                insert.setDouble (4, Math.sqrt(i));
                    insert.executeUpdate ();  Note 6
                }

                System.out.println ("Table " + collectionName + "." + tableName
                    + " has been populated.");
            }

                    catch (Exception e) {
                                System.out.println ();
                System.out.println ("ERROR: " + e.getMessage());
            }

                    finally {

                        try {
                                    if (connection != null)
                        connection.close ();  Note 7
                    }
                            catch (SQLException e) {
                                                    // Ignorare.
                    }
            }

                  System.exit(0);
        }

}
```

1. Questa riga carica l'unità di controllo JDBC IBM Toolbox per Java. Un'unità di controllo JDBC è necessaria per una comunicazione tra JDBC e il database che si sta gestendo.
2. Questa istruzione si collega al database. Verrà visualizzata una finestra che richiede l'ID utente e la parola d'ordine. Viene fornito uno schema predefinito in modo tale che non sia necessario qualificare il nome tabella nelle istruzioni SQL.
3. Queste righe cancellano la tabella se esiste già.
4. Queste righe creano la tabella.
5. Questa riga prepara un'istruzione che inserirà le righe nella tabella. Poiché si andrà ad eseguire questa istruzione più volte, è necessario utilizzare gli indicatori parametro e PreparedStatement.
6. Questo blocco di codice popola la tabella; ogni volta che viene eseguito il loop, inserisce una riga nella tabella.
7. Ora che la tabella è stata creata e popolata, questa istruzione chiude il collegamento con il database.

[Parte successiva]

Esempio: utilizzo delle classi JDBC per creare e popolare una tabella (parte 2 di 2)

[Parte precedente]

Utilizzare quanto segue come esempio per il proprio programma.

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per informazioni legali importanti.

```
///////////////////////////////////////////////////////////////////////////////
//
// Esempio JDBCQuery. Questo programma utilizza l'unità di controllo JDBC IBM Toolbox per Java per
// interrogare una tabella ed emettere il relativo contenuto.
//
// Sintassi del comando:
//    JDBCQuery system collectionName tableName
//
// Ad esempio,
//    JDBCQuery MySystem qiws qcustcdt
//
// Questo sorgente è un esempio di unità di controllo JDBC IBM Toolbox per Java.
//
///////////////////////////////////////////////////////////////////////////////

import java.sql.*;

public class JDBCQuery
{

        // Formattare una stringa in modo che abbia l'ampiezza specificata.
        private static String format (String s, int width)
    {
                String formattedString;

                // La stringa ha un'ampiezza inferiore a quella specificata,
                // quindi è necessario inserire degli spazi.
                if (s.length() < width) {
                        StringBuffer buffer = new StringBuffer (s);
                        for (int i = s.length(); i < width; ++i)
                buffer.append (" ");
                        formattedString = buffer.toString();
        }

                // Altrimenti, è necessario troncare la stringa.
                                else
                        formattedString = s.substring (0, width);

                return formattedString;
    }

       public static void main (String[] parameters)
    {
              // Controllare i parametri di immissione.
                if (parameters.length != 3) {
                     System.out.println("");
                     System.out.println("Usage:");
                     System.out.println("");
            System.out.println("   JDBCQuery system collectionName tableName");
                     System.out.println("");
                     System.out.println("");
                     System.out.println("For example:");
                     System.out.println("");
                     System.out.println("");
            System.out.println("   JDBCQuery mySystem qiws qcustcdt");
                     System.out.println("");
            return;
        }

                 String system = parameters[0];
                String collectionName   = parameters[1];
                String tableName        = parameters[2];

                Connection connection   = null;

                try {
```

```
    DriverManager.registerDriver(new com.ibm.as400.access.AS400JDBCDriver());  Note 1

    // Richiamare un collegamento al database. Poiché non viene fornito
                // un id utente o una parola d'ordine, verrà visualizzata una richiesta.
    connection = DriverManager.getConnection ("jdbc:as400://" + system);
    DatabaseMetaData dmd = connection.getMetaData ();  Note 2

                // Eseguire l'interrogazione.
                Statement select = connection.createStatement ();
    ResultSet rs = select.executeQuery ("SELECT * FROM "
        + collectionName + dmd.getCatalogSeparator() + tableName);  Note 3

    // Richiamare le informazioni sulla serie di risultati. Impostare l'ampiezza
                // della colonna sulla più lunga: lunghezza dell'etichetta
                // o lunghezza dei dati.
                ResultSetMetaData rsmd = rs.getMetaData ();
    int columnCount = rsmd.getColumnCount ();  Note 4
                String[] columnLabels = new String[columnCount];
                int[] columnWidths = new int[columnCount];
                    for (int i = 1; i <= columnCount; ++i) {
                        columnLabels[i-1] = rsmd.getColumnLabel (i);
                        columnWidths[i-1] = Math.max (columnLabels[i-1].length(),
                                    rsmd.getColumnDisplaySize (i));  Note 5
    }

                // Emettere le intestazioni di colonna.
                    for (int i = 1; i <= columnCount; ++i) {
                        System.out.print (format (rsmd.getColumnLabel(i), columnWidths[i-1]));
                 System.out.print("   ");
    }
                    System.out.println ();

                // Emettere una riga con trattini.
                StringBuffer dashedLine;
                    for (int i = 1; i <= columnCount; ++i) {
                        for (int j = 1; j <= columnWidths[i-1]; ++j)
            System.out.print ("-");
                 System.out.print("   ");
    }
                    System.out.println ();

                // Ripetere per le righe nella serie di risultati ed emettere
                // le colonne relative ad ogni riga.
                while (rs.next ()) {
                        for (int i = 1; i <= columnCount; ++i) {
                                String value = rs.getString (i);
                                if (rs.wasNull ())
                value = "<null>";  Note 6
                                System.out.print (format (value, columnWidths[i-1]));
                     System.out.print("   ");
        }
                        System.out.println ();
    }

}

        catch (Exception e) {
                    System.out.println ();
    System.out.println ("ERROR: " + e.getMessage());
}

        finally {

                // Ripulire.
            try {
                        if (connection != null)
                                connection.close ();
```

```
            }
                        catch (SQLException e) {
                                                // Ignorare.
            }
        }

        System.exit (0);
    }

}
```

1. Questa riga carica l'unità di controllo JDBC IBM Toolbox per Java. Una unità di controllo funge da mediatore tra JDBC e il database che si sta gestendo.
2. Questa riga richiama i metadati del collegamento, un oggetto che descrive molte delle caratteristiche del database.
3. Questa istruzione esegue l'interrogazione sulla tabella specificata.
4. Queste righe richiamano le informazioni sulla tabella.
5. Queste righe impostano l'ampiezza colonna sulla lunghezza dell'etichetta o la lunghezza dei dati, quando è più lunga.
6. Questo blocco di codice si ripete su tutte le righe nella tabella e visualizza il contenuto di ogni colonna in ogni riga.

[Parte precedente]

Esempio: visualizzazione di un elenco di lavori server in una GUI

Utilizzare quanto segue come esempio per il proprio programma.

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per informazioni legali importanti.

```
////////////////////////////////////////////////////////////////////////////
//
// Esempio di utilizzo di VJobList vaccess di IBM Toolbox
// per Java.
//
// Questo sorgente è un esempio di "Job List" di IBM Toolbox per Java.
//
////////////////////////////////////////////////////////////////////////////

package examples;  Note 1

import com.ibm.as400.access.*;
import com.ibm.as400.vaccess.*;  Note 2

import javax.swing.*;  Note 3
import java.awt.*;
import java.awt.event.*;

public class GUIExample
{

  public static void main(String[] parameters)  Note 4
  {
          GUIExample example = new GUIExample(parameters);

  }

    public GUIExample(String[] parameters)
  {
     try  Note 5
     {
               // Creare un oggetto AS400.
          //Il nome di sistema è stato passato come primo argomento della riga comandi.
          AS400 system = new AS400 (parameters[0]);  Note 6
```

```
     VJobList jobList = new VJobList (system);  Note 7

            // Creare una frame.
    JFrame frame = new JFrame ("Job List Example");  Note 8

    // Creare un adattatore di finestra di dialogo di errore. Mostrerà qualsiasi errore all'utente.
    ErrorDialogAdapter errorHandler = new ErrorDialogAdapter (frame);  Note 9

            // Creare un pannello explorer per presentare l'elenco di lavori.
    AS400ExplorerPane explorerPane = new AS400ExplorerPane (jobList);  Note 10

    explorerPane.addErrorListener (errorHandler);  Note 11

     // Utilizzare load per caricare le informazioni dal sistema.

    explorerPane.load();  Note 12

     // Quando la frame si chiude, uscire dal programma.
    frame.addWindowListener (new WindowAdapter ()  Note 13
    {
                        public void windowClosing (WindowEvent event)
      {
                  System.exit(0);
      }
    } );

    // Effettuare il layout della frame con il pannello explorer.
    frame.getContentPane().setLayout(new BorderLayout() );
    frame.getContentPane().add("Center", explorerPane);  Note 14

    frame.pack();
    frame.show();  Note 15
    }

                              catch (Exception e)
    {
     e.printStackTrace();  Note 16
    }
    System.exit(0);  Note 17

    }
}
```

1. Questa classe si trova nel pacchetto di esempi. Java utilizza i pacchetti per evitare dei conflitti con i nomi tra i file di classe Java.
2. Questa riga rende tutte le classi IBM Toolbox per Java presenti nel pacchetto Vaccess disponibili per questo programma. Le classi nel pacchetto vaccess hanno il prefisso comune com.ibm.as400.vaccess. Tramite l'utilizzo di un'istruzione Import, il programma richiama il nome invece del pacchetto più il nome. Ad esempio, è possibile fare riferimento alla classe AS400ExplorerPane class utilizzando AS400ExplorerPane, non com.ibm.as400.AS400ExplorerPane.
3. Questa riga rende tutte le JFC (Java Foundation Classes) presenti nel pacchetto Swing disponibili per questo programma. I programmi Java che utilizzano le classi GUI vaccess di IBM Toolbox per Java necessitano di JDK 1.1.2 più Java Swing 1.0.3 della Sun Microsystems, Inc. Swing è disponibile con il JFC 1.1 della Sun.
4. Questa classe dispone di un metodo principale quindi può essere eseguita come un'applicazione. Per richiamare il programma, eseguire "java examples.GUIExample serverName", dove serverName è il nome del server. jt400.zip o jt400.jar devono trovarsi nel percorso classe per fare in modo che questo funzioni.
5. Il codice IBM Toolbox per Java emette delle eccezioni che il programma deve rilevare.
6. La classe AS400 viene utilizzata dall'IBM Toolbox per Java. Questa classe gestisce le informazioni sul collegamento, crea e gestisce i collegamenti socket e invia e riceve i dati. In questo esempio, il programma inoltrerà il nome server all'oggetto AS400.

7. La classe VJobList viene utilizzata dall'IBM Toolbox per Java per rappresentare un elenco di lavori server che è possibile visualizzare in un componente (GUI) Vaccess. Notare che l'oggetto AS400 viene utilizzato per specificare il server su cui risiede l'elenco.
8. Questa riga costituisce una frame o una finestra di livello superiore che verrà utilizzata per visualizzare l'elenco di lavori.
9. ErrorDialogAdapter è un componente GUI di IBM Toolbox per Java che viene creato per visualizzare automaticamente una finestra di dialogo quando si verifica un evento di errore nell'applicazione.
10. Questa riga crea un AS400ExplorerPane, una GUI che rappresenta una gerarchia di oggetti all'interno di una risorsa server. L'AS400ExplorerPane presenta un albero sulla parte sinistra che porta a VJobList e i dettagli della risorsa sulla parte destra. Ciò inizializza solo il pannello in uno stato predefinito e non carica il contenuto di VJobList sul pannello.
11. Questa riga aggiunge l'handler degli errori creato nel passo nove come listener sul componente GUI VJobList.
12. Questa riga carica il contenuto di JobList nell'ExplorerPane. Questo metodo deve essere richiamato esplicitamente per comunicare e caricare informazioni dal server. Ciò fornisce il controllo dell'applicazione quando si verifica la comunicazione con il server. Con ciò è possibile:
    - Caricare il contenuto prima di aggiungere il pannello a una frame. La frame non appare fino a quando non vengono caricate tutte le informazioni, come in questo esempio.
    - Caricare il contenuto dopo aver aggiunto il pannello a una frame e visualizzato tale frame. La frame appare con un "cursore di attesa" e le informazioni vengono inserite non appena vengono caricate.
13. Questa riga aggiunge un listener della finestra in modo tale che l'applicazione termini alla chiusura della frame.
14. Questa riga aggiunge il componente GUI (Graphical user interface) dell'elenco lavori al centro della frame di controllo.
15. Questa riga richiama il metodo di visualizzazione per rendere la finestra visibile all'utente.
16. Le eccezioni IBM Toolbox per Java vengono tradotte in modo tale che il testo apparirà nella lingua della stazione di lavoro. Ad esempio, questo programma visualizza il testo dell'eccezione come errore di elaborazione.
17. L'IBM Toolbox per Java crea i sottoprocessi per eseguire l'attività IBM Toolbox per Java. Se il programma non emette System.exit(0) quando termina, il programma potrebbe non chiudersi correttamente. Ad esempio, se il programma è stato eseguito da una richiesta di dos Windows 95 senza questa riga, non verrà restituita la richiesta comandi alla chiusura del programma.

Esempi: suggerimenti per la programmazione

Questa sezione elenca gli esempi di codice forniti in tutta la documentazione dei suggerimenti per la programmazione di IBM Toolbox per Java.

Gestione dei collegamenti

- Esempio: come stabilire un collegamento al server iSeries con un oggetto CommandCall
- Esempio: come stabilire due collegamenti al server iSeries con un oggetto CommandCall
- Esempio: creazione di oggetti CommandCall e IFSFileInputStream con un oggetto AS400
- Esempio: utilizzo di AS400ConnectionPool per precollegarsi al server iSeries
- Esempio: utilizzo di AS400ConnectionPool per precollegarsi ad un servizio specifico sul server iSeries, quindi riutilizzare il collegamento.

Avvio e chiusura dei collegamenti

- Esempio: come un programma Java si precollega ad un server iSeries
- Esempio: come un programma Java si scollega da un server iSeries

- Esempio: come un programma Java si scollega e si ricollega al server iSeries con disconnectService() e run()
- Esempio: come un programma Java si scollega dal server iSeries e non riesce a collegarsi nuovamente

Eccezioni
- Esempio: utilizzo delle eccezioni

Eventi di errore
- Esempio: gestione degli eventi di errore
- Esempio: definizione di un listener di errori
- Esempio: utilizzo di un handler personalizzato per gestire eventi di errori

Traccia
- Esempio: utilizzo della traccia
- Esempio: utilizzo di setTraceOn()
- Esempio: utilizzo della traccia componente

Ottimizzazione
- Esempio: creazione di due oggetti AS400
- Esempio: utilizzo di un oggetto AS400 per rappresentare un secondo server

Installazione e aggiornamento
- Esempio: utilizzo della classe AS400Toolbox Installer

Il seguente esonero di responsabilità si applica a tutti gli esempi IBM Toolbox per Java:

Esempi: ToolboxME per iSeries

Questa sezione elenca gli esempi di codice che sono forniti in tutta la documentazione IBM ToolboxME per iSeries.

- "Esempio ToolboxME per iSeries: JdbcDemo.java" a pagina 367
- "Esempio: utilizzo di ToolboxME per iSeries, MIDP e JDBC" a pagina 692
- "Esempio: utilizzo di ToolboxME per iSeries, MID e IBM Toolbox per Java" a pagina 700

Il seguente esonero di responsabilità si applica a tutti gli esempi IBM Toolbox per Java:

L'IBM fornisce una licenza non esclusiva per utilizzare tutti gli esempi del codice di programmazione da cui creare funzioni simili personalizzate, in base a richieste specifiche.

Questo codice di esempio è fornito dall'IBM con la sola funzione illustrativa. Questi esempi non sono stati interamente testati in tutte le condizioni. IBM, perciò, non fornisce nessun tipo di garanzia o affidabilità implicita, rispetto alla funzionalità o alle funzioni di questi programmi.

Tutti i programmi qui contenuti vengono forniti all'utente ″COSI' COME SONO″ senza garanzie di alcun tipo. Le garanzie implicite di non contraffazione, commerciabilità e adeguatezza a scopi specifici sono espressamente vietate.

Esempio: utilizzo di ToolboxME per iSeries, MIDP e JDBC

Il seguente sorgente illustra un modo in cui l'applicazione ToolboxME per iSeries può utilizzare MIDP (Mobile Information Device Profile) e JDBC per accedere ad un database e memorizzare le informazioni non in linea.

Questo esempio dimostra come un agente immobiliare potrebbe essere in grado di visualizzare e fare un'offerta per le proprietà attualmente in vendita. L'agente utilizza un'unità Tier0 per accedere alle informazioni relative alle proprietà, che sono memorizzate nel database del server iSeries.

Una volta creato come programma di lavoro, il codice di esempio sottostante si effettua il collegamento ad un database creato a tale scopo.

Per creare una versione di lavoro del codice sorgente ed ottenere il sorgente per la creazione ed il popolamento del database richiesto, è necessario scaricare l'esempio.E' possibile anche esaminare le istruzioni per la creazione e l'esecuzione del programma di esempio.

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per informazioni legali importanti.

```
////////////////////////////////////////////////////////////////////////////
//
// Esempio ToolboxME per iSeries. Questo programma è un MIDlet di esempio che mostra come
// codificare un'applicazione JdbcMe per il profilo MIDP. Fare riferimento ai
// metodi startApp, pauseApp, destroyApp e commandAction per visualizzare come gestisce
// ogni transizione richiesta.
//
////////////////////////////////////////////////////////////////////////////

import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
import java.sql.*;
import javax.microedition.rms.*;

import com.ibm.as400.micro.*;


public class JdbcMidpBid extends MIDlet implements CommandListener
{
        private static int BID_PROPERTY = 0;
        private Display   display;

    private TextField urlText = new TextField("urltext",
                                          "jdbc:as400://mySystem;user=myUid;password=myPwd;",
                                          65,
                                          TextField.ANY);
    private TextField jdbcmeText = new TextField("jdbcmetext", "meserver=myMEServer", 40, TextField.ANY);
    private TextField jdbcmeTraceText = new TextField("jdbcmetracetext", "0", 10, TextField.ANY);
    private final static String GETBIDS = "No bids are available, select here to download bids";
    private List      main = new List("JdbcMe Bid Demo", Choice.IMPLICIT);
        private List      listings = null;
        private Form      aboutBox;
```

```
    private Form      bidForm;
    private Form      settingsForm;
    private int       bidRow = 0;
    private String    bidTarget = null;
    private String    bidTargetKey = null;
private TextField bidText = new TextField("bidtext", "", 10, TextField.NUMERIC);
    private Form      errorForm = null;

private Command exitCommand   = new Command("Exit", Command.SCREEN, 0);
private Command backCommand   = new Command("Back", Command.SCREEN, 0);
private Command cancelCommand = new Command("Cancel", Command.SCREEN, 0);
private Command goCommand     = new Command("Go", Command.SCREEN, 1);
    private Displayable   onErrorGoBackTo = null;

/*
      * Construct a new JdbcMidpBid.
 */
    public JdbcMidpBid()
{
            display = Display.getDisplay(this);
}

/**
      * Show the main screen
 */
    public void startApp()
{
    main.append("Show Bids", null);
    main.append("Get New Bids", null);
    main.append("Settings", null);
    main.append("About", null);
            main.addCommand(exitCommand);
            main.setCommandListener(this);

                        display.setCurrent(main);
}

    public void commandAction(Command c, Displayable s)
{
            // L'elaborazione di tutti gli exitCommand è la stessa.
            if (c == exitCommand)
    {
                    destroyApp(false);
                    notifyDestroyed();
        return;
    }
            if (s instanceof List)
    {
                    List     current = (List)s;

                    // Si è verificata un'azione sulla pagina principale
                    if (current == main)
        {
                            int   idx = current.getSelectedIndex();
                            switch (idx)
            {
                            case 0:     // Visualizzare le offerte correnti
                        showBids();
                                break;
                            case 1:     // Richiamare le nuove offerte
                                getNewBids();
                                break;
                            case 2:     // Impostazioni
                                    doSettings();
                                break;
                            case 3:     // Info su
                                    aboutBox();
```

```
                                break;
                        default :
                                break;
        }
        return;
                } // current == main

                // Si è verificata un'azione sulla pagina delle elencazioni
                if (current == listings)
    {
                        if (c == backCommand)
        {
                                display.setCurrent(main);
            return;
        }
                        if (c == List.SELECT_COMMAND)
        {
                                int idx = listings.getSelectedIndex();
                                String stext = listings.getString(idx);
                                if (stext.equals(GETBIDS))
            {
                                        getNewBids();
                return;
            }
                                int   commaIdx = stext.indexOf(',');
                                bidTargetKey = stext.substring(0, commaIdx);
            bidTarget = stext.substring(commaIdx+1) + "\n";
            // Inoltre tenere traccia di quale riga della serie di risultati fuori linea
            // si tratta. Risulta essere uguale all'indice
            // nell'elenco.
                                bidRow = idx;

                                bidOnProperty();
        }
                } // current == listings
    return;
        } // instanceof List
        if (s instanceof Form)
{
                Form     current = (Form)s;
                if (current == errorForm)
    {
                        if (c == backCommand)
                                display.setCurrent(onErrorGoBackTo);

        return;
                } // errorForm
                if (current == settingsForm)
    {
                        if (c == backCommand)
        {
                                // Terminate le impostazioni.
                                display.setCurrent(main);
                                settingsForm = null;
            return;
        }
                } // settingsForm
                if (current == aboutBox)
    {
                        if (c == backCommand)
        {
                                // Completata la casella Info su.
                                display.setCurrent(main);
                                aboutBox = null;
            return;
        }
    }
```

```
                    if (current == bidForm)
        {
                            if (c == cancelCommand)
            {
                        display.setCurrent(listings);
                                        bidForm = null;
                return;
            }
                            if (c == goCommand)
            {
                                    submitBid();
                                    if (display.getCurrent() != bidForm)
                {
                                            // Se non si è più posizionati su
                                            // bidForm, verrà eliminato.
                                            bidForm = null;
                }
                return;
            }
            return;
                    } // current == bidForm
            } // instanceof Form
}

    public void aboutBox()
{
    aboutBox = new Form("aboutbox");
    aboutBox.setTitle("About");
    aboutBox.append(new StringItem("", "Midp RealEstate example for JdbcMe "));
            aboutBox.addCommand(backCommand);
            aboutBox.setCommandListener(this);
            display.setCurrent(aboutBox);
}

/**
      *  The settings form.
 */
    public void doSettings()
{
    settingsForm = new Form("settingsform");
    settingsForm.setTitle("Settings");
    settingsForm.append(new StringItem("", "DB URL"));
            settingsForm.append(urlText);
    settingsForm.append(new StringItem("", "JdbcMe server"));
            settingsForm.append(jdbcmeText);
    settingsForm.append(new StringItem("", "Trace"));

            settingsForm.addCommand(backCommand);
            settingsForm.setCommandListener(this);
            display.setCurrent(settingsForm);
}


/**
      * Show the bid screen for the bid target
      * that we selected.
 */
    public void bidOnProperty()
{
    StringItem item = new StringItem("", bidTarget);

    bidText = new TextField("bidtext", "", 10, TextField.NUMERIC);
    bidText.setString("");

    bidForm = new Form("bidform");
    bidForm.setTitle("Submit a bid for:");
            BID_PROPERTY = 0;
```

```
            bidForm.append(item);
    bidForm.append(new StringItem("", "Your bid:"));
            bidForm.append(bidText);
            bidForm.addCommand(cancelCommand);
            bidForm.addCommand(goCommand);
            bidForm.setCommandListener(this);
            display.setCurrent(bidForm);
}

/**
      * Update the listings card with the
 * current list of bids that we are interested in.
 */
    public void getNewBids()
{
            // Ripristinare la vecchia elencazione
            listings = null;
    listings = new List("JdbcMe Bids", Choice.IMPLICIT);
            java.sql.Connection     conn = null;
            Statement               stmt = null;
                                try
    {
        conn = DriverManager.getConnection(urlText.getString() + ";" + jdbcmeText.getString());

                    stmt = conn.createStatement();

        // Poiché non si desidera che l'istruzione preparata persista,
        // un'istruzione normale è davvero la scelta migliore in questo ambiente.
        String sql = "select mls, address, currentbid from qjdbcme.realestate where currentbid <> 0";

        boolean results =((JdbcMeStatement)stmt).executeToOfflineData(sql,
                                                                      "JdbcMidpBidListings",
                                                                      0,
                                                                      0);
                    if (results)
        {
                        setupListingsFromOfflineData();
        }
                                else
        {
            listings.append("No bids found", null);
                    listings.addCommand(backCommand);
                    listings.setCommandListener(this);
        }
    }
                                catch (Exception e)
    {
        // Attualmente non viene richiamata alcuna elencazione valida, quindi consentire
        // la reimpostazione su vuoto.
        listings = new List("JdbcMe Bids", Choice.IMPLICIT);
                        listings.append(GETBIDS, null);
                listings.addCommand(backCommand);
                listings.setCommandListener(this);

                    // Ritornare alla principale dopo aver visualizzato l'errore.
                    showError(main, e);
        return;
    }
            finally
    {
                    if (conn != null)
        {
                                    try
            {
                                conn.close();
            }
                                    catch (Exception e)
```

```
            {
            }
        }
                conn = null;
                stmt = null;
    }
            showBids();
}

    public void setupListingsFromOfflineData()
{
            // Ignorare le prime quattro righe nella memoria del record
            // (eyecatcher, version, num columns, sql column
            // types)
            // ed ogni riga successiva nella memoria del record è
    // una singola colonna. L'interrogazione restituisce 3 colonne
    // che verranno restituite concatenate come una singola stringa.
                ResultSet         rs = null;
            listings.addCommand(backCommand);
            listings.setCommandListener(this);
                            try
    {
                int                     i = 5;
                int                     max = 0;
                StringBuffer   buf  = new StringBuffer(20);

                        // Creatore e dbtype inutilizzati in MIDP
        rs = new JdbcMeOfflineResultSet("JdbcMidpBidListings", 0, 0);
                if (rs == null)
        {
                        // Nuove elencazioni...
            listings = new List("JdbcMe Bids", Choice.IMPLICIT);
                        listings.append(GETBIDS, null);
                listings.addCommand(backCommand);
                listings.setCommandListener(this);
            return;
        }

                i = 0;
                String      s = null;
                while (rs.next())
        {
                        ++i;

                        s = rs.getString(1);
                        buf.append(s);

            buf.append(",");
                        s = rs.getString(2);
                        buf.append(s);

            buf.append(", $");
                        s = rs.getString(3);
                        buf.append(s);

                        listings.append(buf.toString(), null);
                        buf.setLength(0);
        }

                if (i == 0)
        {
            listings.append("No bids found", null);
            return;
        }
    }
                            catch (Exception e)
    {
```

```
        // Attualmente non viene richiamata alcuna elencazione valida, quindi consentire
        // la reimpostazione su vuoto.
        listings = new List("JdbcMe Bids", Choice.IMPLICIT);
                        listings.append(GETBIDS, null);
                listings.addCommand(backCommand);
                listings.setCommandListener(this);

                    // Ritornare alla principale dopo aver visualizzato l'errore.
                    showError(main, e);
        return;
    }
            finally
    {
                            if (rs != null)
        {
                                        try
            {
                                        rs.close();
            }
                                        catch (Exception e)
            {
            }
                            rs = null;
        }
                    System.gc();
    }
}

/**
      * Update the listings card with the
 * current list of bids that we are interested in.
 */
    public void submitBid()
{
            java.sql.Connection     conn = null;
            Statement               stmt = null;
                                try
    {
        conn = DriverManager.getConnection(urlText.getString() + ";" + jdbcmeText.getString());

                    stmt = conn.createStatement();

        // Poiché non si desidera che l'istruzione preparata persista,
        // un'istruzione normale è davvero la scelta migliore in questo ambiente.
                    StringBuffer   buf = new StringBuffer(100);
        buf.append("Update QJdbcMe.RealEstate Set CurrentBid = ");
                    buf.append(bidText.getString());
        buf.append(" Where MLS = '");
                    buf.append(bidTargetKey);
        buf.append("' and CurrentBid < ");
                    buf.append(bidText.getString());
                    String      sql = buf.toString();

                    int   updated = stmt.executeUpdate(sql);
                    if (updated == 1)
        {
                            // OFFERTA Accettata.
                            String oldS = listings.getString(bidRow);
                            int   commaIdx = bidTarget.indexOf(',');
                            String bidAddr = bidTarget.substring(0, commaIdx);

            String newS = bidTargetKey + "," + bidAddr + ", $" + bidText.getString();

                            ResultSet         rs = null;
                                        try
            {
                                    // Creatore e dbtype inutilizzati in MIDP
```

```
                rs = new JdbcMeOfflineResultSet("JdbcMidpBidListings", 0, 0);
                                    rs.absolute(bidRow+1);
                                    rs.updateString(3, bidText.getString());
                                        rs.close();
            }
                                        catch (Exception e)
            {
                                    if (rs != null)
                                            rs.close();
            }

                                // Inoltre aggiornare l'elenco attivo di tale serie di risultati.
                                listings.set(bidRow, newS, null);
                    display.setCurrent(listings);
                                conn.commit();
        }
                                    else
        {
                                conn.rollback();
            throw new SQLException("Failed to bid, someone beat you to it");
        }
    }
            catch (SQLException e)
    {
                    // Ritornare al modulo offerta dopo la visualizzazione dell'errore.
                    showError(bidForm, e);
        return;
    }
            finally
    {
                    if (conn != null)
        {
                                        try
            {
                                    conn.close();
            }
                                        catch (Exception e)
            {
            }
        }
                    conn = null;
                    stmt = null;
    }

            // Uscire senza eccezioni, quindi visualizzare le offerte correnti
            showBids();
}

/**
      * Show an error condition.
 */
    public void showError(Displayable d, Exception e)
{
            String s = e.toString();

            onErrorGoBackTo = d;
    errorForm = new Form("Error");
    errorForm.setTitle("SQL Error");
    errorForm.append(new StringItem("", s));
            errorForm.addCommand(backCommand);
            errorForm.setCommandListener(this);
            display.setCurrent(errorForm);
}

/**
      * Show the current bids.
 */
```

```
        public void showBids()
    {
                if (listings == null)
        {
                        // Se non si dispone di elencazioni correnti, consentirne
                        // l'impostazione.
            listings = new List("JdbcMe Bids", Choice.IMPLICIT);
                        setupListingsFromOfflineData();
        }
                display.setCurrent(listings);
    }

    /**
     * Time to pause, free any space we do not need right now.
     */
        public void pauseApp()
    {
                display.setCurrent(null);
    }

    /**
          * Destroy must cleanup everything.
     */
        public void destroyApp(boolean unconditional)
    {
    }
}
```

Esempio: utilizzo di ToolboxME per iSeries, MID e IBM Toolbox per Java

Il seguente sorgente illustra un modo in cui l'applicazione ToolboxME per iSeries può utilizzare MIDP (Mobile Information Device Profile) e IBM Toolbox per Java per accedere ai dati e ai servizi su un server iSeries.

Questo esempio fornisce una dimostrazione di ognuna delle funzioni create nel supporto IBM Toolbox per Java 2 Micro Edition. Questa applicazione presenta differenti pagine o schemi che illustrano alcune delle molte modalità in cui l'unità Tier0 può utilizzare queste funzioni.

Una volta creato come programma di lavoro, il codice di esempio sottostante utilizza un file PCML (Program Call Markup Language) per eseguire i comandi sul server iSeries.

Per creare una versione di lavoro del codice sorgente ed ottenere il sorgente PCML richiesto per eseguire i comandi del server, è necessario scaricare l'esempio.E' possibile anche esaminare le istruzioni per la creazione e l'esecuzione del programma di esempio.

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per informazioni legali importanti.

```
////////////////////////////////////////////////////////////////////////////
//
// Esempio ToolboxME per iSeries. Questo programma è un esempio che mostra come
// ToolboxME per iSeries può utilizzare PCML per accedere a dati e servizi su un
// server iSeries.
//
// Questa applicazione richiede che il file qsyrusri.pcml sia presente in
// CLASSPATH di MEServer.
//
////////////////////////////////////////////////////////////////////////////

import java.io.*;
import java.sql.*;
import java.util.Hashtable;

import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
```

```
import javax.microedition.rms.*;

import com.ibm.as400.micro.*;


public class ToolboxMidpDemo extends MIDlet implements CommandListener
{
        private Display   display_;

        // Un oggetto sistema ToolboxME.
        private AS400 system_;

    private List      main_ = new List("ToolboxME MIDP Demo", Choice.IMPLICIT);

        // Creare un modulo per ogni componente.
        private Form      signonForm_;
        private Form      cmdcallForm_;
        private Form      pgmcallForm_;
        private Form      dataqueueForm_;
        private Form      aboutForm_;

        // Testo visibile per ogni componente.
    static final String SIGN_ON       = "SignOn";
    static final String COMMAND_CALL  = "CommandCall";
    static final String PROGRAM_CALL  = "ProgramCall";
    static final String DATA_QUEUE    = "DataQueue";
    static final String ABOUT         = "About";

    static final String NOT_SIGNED_ON = "Not signed on.";
    static final String DQ_READ       = "Read";
    static final String DQ_WRITE      = "Write";

        // Un ticker per visualizzare lo stato del collegamento.
        private Ticker    ticker_ = new Ticker(NOT_SIGNED_ON);

        // Comandi che è possibile eseguire.
    private static final Command actionExit_   = new Command("Exit", Command.SCREEN, 0);
    private static final Command actionBack_   = new Command("Back", Command.SCREEN, 0);
    private static final Command actionGo_     = new Command("Go", Command.SCREEN, 1);
    private static final Command actionClear_  = new Command("Clear", Command.SCREEN, 1);
    private static final Command actionRun_    = new Command("Run", Command.SCREEN, 1);
        private static final Command actionSignon_ = new Command(SIGN_ON, Command.SCREEN, 1);
    private static final Command actionSignoff_= new Command("SignOff", Command.SCREEN, 1);

        private Displayable   onErrorGoBackTo_;
    // il modulo da restituire una volta terminata la visualizzazione del modulo errori

        // TextField per il modulo SignOn.
    private TextField signonSystemText_ = new TextField("System", "rchasdm3", 20, TextField.ANY);
    private TextField signonUidText_ = new TextField("UserId", "JAVA", 10, TextField.ANY);
    // TBD temporary
    private TextField signonPwdText_ = new TextField("Password", "JTEAM1", 10, TextField.PASSWORD);
    private TextField signonServerText_ = new TextField("MEServer", "localhost", 10, TextField.ANY);
    private StringItem signonStatusText_ = new StringItem("Status", NOT_SIGNED_ON);

        // TextField per il modulo CommandCall.
    // TBD: max size; TBD: TextBox???
    private TextField cmdText_ = new TextField("Command", "CRTLIB FRED", 256, TextField.ANY);
    private StringItem cmdMsgText_ = new StringItem("Messages", null);
    private StringItem cmdStatusText_ = new StringItem("Status", null);

        // TextField per il modulo ProgramCall.
    private StringItem pgmMsgDescription_ = new StringItem("Messages", null);
    private StringItem pgmMsgText_ = new StringItem("Messages", null);

        // TextField per il modulo DataQueue.
    private TextField dqInputText_ = new TextField("Data to write", "Hi there", 30, TextField.ANY);
```

```
private StringItem dqOutputText_ = new StringItem("DQ contents", null);
private ChoiceGroup dqReadOrWrite_ = new ChoiceGroup("Action",
                                                     Choice.EXCLUSIVE,
                                                     new String[] { DQ_WRITE, DQ_READ},
                                                                                          null);
private StringItem dqStatusText_ = new StringItem("Status", null);


/**
      * Crea un nuovo ToolboxMidpDemo.
 **/
    public ToolboxMidpDemo()
{
            display_ = Display.getDisplay(this);
    // Nota: nel demo basato su KVM è stato utilizzato TabbedPane per il pannello principale.
    // MIDP non dispone di una classe simile, pertanto verrà utilizzata una List.
}

/**
      * Mostra lo schermo principale.
      * Implementa il metodo astratto della classe Midlet.
 **/
    protected void startApp()
{
            main_.append(SIGN_ON, null);
            main_.append(COMMAND_CALL, null);
            main_.append(PROGRAM_CALL, null);
            main_.append(DATA_QUEUE, null);
            main_.append(ABOUT, null);

            main_.addCommand(actionExit_);
            main_.setCommandListener(this);

                display_.setCurrent(main_);
}

    // Implementa metodo dell'interfaccia CommandListener.
    public void commandAction(Command action, Displayable dsp)
{
            // Tutta l'elaborazione 'exit' e 'back' è uguale.
            if (action == actionExit_)
    {
                    destroyApp(false);

                    notifyDestroyed();
    }
            else if (action == actionBack_)
    {
                    // Ritornare al menu principale.
                    display_.setCurrent(main_);
    }
            else if (dsp instanceof List)
    {
                    List current = (List)dsp;

                    // Si è verificata un'azione sulla pagina principale
                    if (current == main_)
        {
                            int   idx = current.getSelectedIndex();

                            switch (idx)
            {
                            case 0:     // SignOn
                                    showSignonForm();
                                        break;
                            case 1:     // CommandCall
                                    showCmdForm();
```

```
                                break;
                        case 2:     // ProgramCall
                                showPgmForm();
                                break;
                        case 3:     // DataQueue
                                showDqForm();
                                break;
                        case 4:     // Info su
                                showAboutForm();
                                break;
                        default:    // Nessuna delle precedenti
            feedback("Internal error: Unhandled selected index in main: " + idx,
                     AlertType.ERROR);
                                break;
        }
                } // current == main
                            else
        feedback("Internal error: The Displayable object is a List but is not main_.",
                 AlertType.ERROR);
        } // instanceof List
        else if (dsp instanceof Form)
{
                Form current = (Form)dsp;

                if (current == signonForm_)
    {
                        if (action == actionSignon_)
        {
            // Creare un oggetto sistema ToolboxME.
            system_ = new AS400(signonSystemText_.getString(),
                                signonUidText_.getString(),
                                signonPwdText_.getString(),
                                signonServerText_.getString());

                                        try
            {
                // Collegarsi all'iSeries.
                                        system_.connect();

                // Impostare il testo sullo stato del collegamento.
                signonStatusText_.setText("Signed on.");

                // Visualizzare una finestra di dialogo di conferma del
                // collegamento dell'utente.
                feedback("Successfully signed on.", AlertType.INFO, main_);

                // Sostituire il pulsante SignOn con SignOff.
                signonForm_.removeCommand(actionSignon_);
                signonForm_.addCommand(actionSignoff_);

                // Aggiornare il ticker.
                ticker_.setString("... Signed on to '" +
                                  signonSystemText_.getString() + "' as '" +
                                  signonUidText_.getString() + "' via '" +
                                  signonServerText_.getString() + "' ... ");
            }
                                        catch (Exception e)
            {
                        e.printStackTrace();

                // Impostare il testo sullo stato del collegamento.
                signonStatusText_.setText(NOT_SIGNED_ON);

                feedback("Signon failed.  " + e.getMessage(), AlertType.ERROR);
            }
        }
                        else if (action == actionSignoff_)
```

```
    {
                        if (system_ == null)
            feedback("Internal error: System is null.", AlertType.ERROR);
                         else
        {
                                  try
            {
                // Scollegarsi dall'iSeries.
                         system_.disconnect();
                                     system_ = null;

                // Impostare il testo sullo stato del collegamento.
                signonStatusText_.setText(NOT_SIGNED_ON);

                // Visualizzare una finestra di dialogo di conferma che l'utente non è più collegato.
                feedback("Successfully signed off.", AlertType.INFO, main_);

                // Sostituire il pulsante SignOff con SignOn.
                                     signonForm_.removeCommand(actionSignoff_);
                        signonForm_.addCommand(actionSignon_);

                // Aggiornare il ticker.
                                     ticker_.setString(NOT_SIGNED_ON);
            }
                                  catch (Exception e)
            {
                                  feedback(e.toString(), AlertType.ERROR);

                    e.printStackTrace();

                signonStatusText_.setText("Error.");

                feedback("Error during signoff.", AlertType.ERROR);
            }
        }
    }
                ;  // Nessuna delle precedenti
    {
        feedback("Internal error: Action is not recognized.", AlertType.INFO);
    }
            } // signonForm_
            else if (current == cmdcallForm_)
{
                    if (action == actionRun_)
    {
        // Se l'utente non si è collegato, visualizzare un avviso.
                        if (system_ == null)
        {
                                feedback(NOT_SIGNED_ON, AlertType.ERROR);
                    return;
        }

        // Richiamare il comando che l'utente ha immesso nell'unità wireless.
                        String cmdString = cmdText_.getString();

        // Se il comando non è stato specificato, visualizzare un avviso.
                        if (cmdString == null || cmdString.length() == 0)
            feedback("Specify command.", AlertType.ERROR);
                         else
        {
                                  try
            {
                // Eseguire il comando.
            String[] messages = CommandCall.run(system_, cmdString);

                StringBuffer status = new StringBuffer("Command completed with ");
```

```
                // Controllare per verificare se vi siano messaggi.
                                        if (messages.length == 0)
                {
                    status.append("no returned messages.");

                                        cmdMsgText_.setText(null);

                    cmdStatusText_.setText("Command completed successfully.");
                }
                                        else
                {
                                                if (messages.length == 1)
                        status.append("1 returned message.");
                                        else
                        status.append(messages.length + " returned messages.");

                    // Se vi sono messaggi, visualizzare solo il primo.
                    cmdMsgText_.setText(messages[0]);

                    cmdStatusText_.setText(status.toString());
                }

                                        repaint();
            }
                                        catch (Exception e)
            {
                                        feedback(e.toString(), AlertType.ERROR);

                    e.printStackTrace();

                feedback("Error when running command.", AlertType.ERROR);
            }
        }
    }
                else if (action == actionClear_)
    {
        // Eliminare il contenuto del testo del comando e dei messaggi.
        cmdText_.setString("");

                        cmdMsgText_.setText(null);

                        cmdStatusText_.setText(null);

                            repaint();
    }
                else  // Nessuna delle precedenti
    {
        feedback("Internal error: Action is not recognized.", AlertType.INFO);
    }
            } // cmdcallForm_
            else if (current == pgmcallForm_)
{
                    if (action == actionRun_)
    {
        // Se l'utente non si è collegato prima di effettuare una chiamata al programma,
        // visualizzare un avviso.
                        if (system_ == null)
        {
                                feedback(NOT_SIGNED_ON, AlertType.ERROR);
                    return;
        }

                        pgmMsgText_.setText(null);

        // Consultare l'esempio PCML presente nelle informazioni relative a IBM Toolbox per Java.
        String pcmlName = "qsyrusri.pcml"; // The PCML file we want to use.
           String apiName = "qsyrusri";
```

```
// Creare una hashtable che contenga i parametri di immissione per la chiamata al programma.
Hashtable parmsToSet = new Hashtable(2);
parmsToSet.put("qsyrusri.receiverLength", "2048");
parmsToSet.put("qsyrusri.profileName", signonUidText_.getString().toUpperCase());

// Creare una schiera di stringhe che contenga i parametri di emissione da richiamare.
String[] parmsToGet = { "qsyrusri.receiver.userProfile",
                        "qsyrusri.receiver.previousSignonDate",
                        "qsyrusri.receiver.previousSignonTime",
                        "qsyrusri.receiver.daysUntilPasswordExpires"};

// Una schiera di stringhe contenente le descrizioni dei parametri da visualizzare.
String[] displayParm = { "Profile",
                         "Last signon Date",
                         "Last signon Time",
                         "Password Expired (days)"};

                            try
{
    // Eseguire il programma.
    String[] valuesToGet = ProgramCall.run(system_,
                                           pcmlName,
                                           apiName,
                                           parmsToSet,
                                           parmsToGet);

    // Creare uno StringBuffer ed aggiungere ogni parametro richiamato.
    StringBuffer txt = new StringBuffer();
    txt.append(displayParm[0] + ": " + valuesToGet[0] + "\n");

    char[] c = valuesToGet[1].toCharArray();
    txt.append(displayParm[1] + ": " + c[3] + c[4] + "/" +
               c[5] + c[6] + "/" + c[1] + c[2] + "\n");

    char[] d = valuesToGet[2].toCharArray();
    txt.append(displayParm[2] + ": " + d[0] + d[1] + ":" + d[2] + d[3] + "\n");
    txt.append(displayParm[3] + ": " + valuesToGet[3] + "\n");

    // Impostare il testo visualizzabile dei risultati della chiamata al programma.
    pgmMsgText_.setText(txt.toString());

    StringBuffer status = new StringBuffer("Program completed with ");

                            if (valuesToGet.length == 0)
    {
        status.append("no returned values.");

                                    feedback(status.toString(), AlertType.INFO);
    }
                            else
    {
                                    if (valuesToGet.length == 1)
            status.append("1 returned value.");
                                else
            status.append(valuesToGet.length + " returned values.");

                                    feedback(status.toString(), AlertType.INFO);
    }
}
                            catch (Exception e)
{
                            feedback(e.toString(), AlertType.ERROR);

            e.printStackTrace();

    feedback("Error when running program.", AlertType.ERROR);
```

```
        }
    }
                else if (action == actionClear_)
    {
        // Eliminare il contenuto dei risultati della chiamata al programma.
                        pgmMsgText_.setText(null);

                            repaint();
    }
            }  // pgmcallForm_
            else if (current == dataqueueForm_)  // DataQueue
{
                    if (action == actionGo_)
    {
        // Se l'utente non si è collegato prima di eseguire azioni DataQueue,
        // verrà visualizzato un messaggio di avviso.
                        if (system_ == null)
        {
                                feedback(NOT_SIGNED_ON, AlertType.ERROR);

                    return;
        }

        // Creare una libreria in cui creare la coda dati
                                try
        {
            CommandCall.run(system_, "CRTLIB FRED");
        }
                                catch (Exception e)
        {
        }

        // Eseguire un comando per creare una coda dati.
                                try
        {
            CommandCall.run(system_, "CRTDTAQ FRED/MYDTAQ MAXLEN(2000)");
        }
                                catch (Exception e)
        {
            feedback("Error when creating data queue.  " + e.getMessage(),
                     AlertType.WARNING);
        }

                                try
        {
            // Visualizzare quale azione è stata selezionata (Read o Write).
            if (dqReadOrWrite_.getString(dqReadOrWrite_.getSelectedIndex()).equals(DQ_WRITE))
            {
                // Scrivere
                                dqOutputText_.setText(null);

                // Richiamare dall'immissione dell'unità senza fili il testo da scrivere
                // nella coda dati.
                                    if (dqInputText_.getString().length() == 0)
                    dqStatusText_.setText("No data specified.");
                                else
                {
                    // Scrivere nella coda dati.
                    DataQueue.write(system_,
                                "/QSYS.LIB/FRED.LIB/MYDTAQ.DTAQ",
                                dqInputText_.getString().getBytes() );

                                            dqInputText_.setString(null);

                    // Visualizzare lo stato.
                    dqStatusText_.setText("The 'write' operation completed.");
                }
```

```
                    }
                                        else  // Leggere
                    {
                        // Leggere dalla coda dati.
                        byte[] b = DataQueue.readBytes(system_, "/QSYS.LIB/FRED.LIB/MYDTAQ.DTAQ");

                        // Stabilire se la coda dati conteneva o meno delle voci
                        // e visualizzare il messaggio appropriato.
                                            if (b == null)
                        {
                            dqStatusText_.setText("No dataqueue entries are available.");

                                        dqOutputText_.setText(null);
                        }
                                            else if (b.length == 0)
                        {
                            dqStatusText_.setText("Dataqueue entry has no data.");

                                        dqOutputText_.setText(null);
                        }
                                        else
                        {
                            dqStatusText_.setText("The 'read' operation completed.");

                                                dqOutputText_.setText(new String(b));
                        }
                    }

                                        repaint();
                }
                                        catch (Exception e)
                {
                            e.printStackTrace();

                                        feedback(e.toString(), AlertType.ERROR);

                    feedback("Error when running command.  " + e.getMessage(), AlertType.ERROR);
                }
                            }  // actionGo_
                            else if (action == actionClear_)
            {
                // Eliminare il contenuto del modulo coda dati.
                dqInputText_.setString("");

                dqOutputText_.setText(null);

                dqReadOrWrite_.setSelectedFlags(new boolean[] { true, false});

                dqStatusText_.setText(null);

                repaint();
            }
            else  // Nessuna delle precedenti
            {
                feedback("Internal error: Action is not recognized.", AlertType.INFO);
            }
                    } // dataqueueForm_
        else if (current == aboutForm_)  // "About".
        {
            // Non si sarebbe dovuti mai arrivare a questo punto, poiché l'unico pulsante è "Indietro".
                    }  // Nessuno dei procedenti.
                               else
            feedback("Internal error: Form is not recognized.", AlertType.ERROR);
            } // instanceof Form
                            else
        feedback("Internal error: Displayable object not recognized.", AlertType.ERROR);
}
```

```
/**
 *  Visualizza il modulo "About".
 **/
    private void showAboutForm()
{
            // Se il modulo info su è nullo, crearlo e accodarlo.
            if (aboutForm_ == null)
    {
                    aboutForm_ = new Form(ABOUT);
        aboutForm_.append(new StringItem(null,
                            "This is a MIDP example application that uses the " +
                            "IBM Toolbox for Java Micro Edition (ToolboxME)."));

                    aboutForm_.addCommand(actionBack_);
                    aboutForm_.setCommandListener(this);
    }

            display_.setCurrent(aboutForm_);
}



/**
 *  Visualizza il modulo "SignOn".
 **/
    private void showSignonForm()
{
            // Creare un modulo per il collegamento.
            if (signonForm_ == null)
    {
                    signonForm_ = new Form(SIGN_ON);
                    signonForm_.append(signonSystemText_);
                    signonForm_.append(signonUidText_);
                    signonForm_.append(signonPwdText_);
                    signonForm_.append(signonServerText_);
                    signonForm_.append(signonStatusText_);
                    signonForm_.addCommand(actionBack_);
                    signonForm_.addCommand(actionSignon_);
                    signonForm_.setCommandListener(this);
                    signonForm_.setTicker(ticker_);
    }

            display_.setCurrent(signonForm_);
}



/**
 *  Visualizza il modulo "CommandCall".
 **/
    private void showCmdForm()
{
            // Creare il modulo per la chiamata al comando
            if (cmdcallForm_ == null)
    {
                    cmdcallForm_ = new Form(COMMAND_CALL);
                    cmdcallForm_.append(cmdText_);
                    cmdcallForm_.append(cmdMsgText_);
                    cmdcallForm_.append(cmdStatusText_);
                    cmdcallForm_.addCommand(actionBack_);
                    cmdcallForm_.addCommand(actionClear_);
                    cmdcallForm_.addCommand(actionRun_);
                    cmdcallForm_.setCommandListener(this);
                    cmdcallForm_.setTicker(ticker_);
```

```
    }

            display_.setCurrent(cmdcallForm_);
}



/**
 *  Visualizza il modulo "ProgramCall".
 **/
    private void showPgmForm()
{
            // Creare il modulo per la chiamata al programma
            if (pgmcallForm_ == null)
    {
                    pgmcallForm_ = new Form(PROGRAM_CALL);
        pgmcallForm_.append(new StringItem(null,
                                           "This calls the Retrieve User Information (QSYRUSRI) " +
                                           "API, and returns information about the current " +
                                           "user profile."));
                    pgmcallForm_.append(pgmMsgText_);
                    pgmcallForm_.addCommand(actionBack_);
                    pgmcallForm_.addCommand(actionClear_);
                    pgmcallForm_.addCommand(actionRun_);
                    pgmcallForm_.setCommandListener(this);
                    pgmcallForm_.setTicker(ticker_);
    }

            display_.setCurrent(pgmcallForm_);
}


/**
 *  Visualizza il modulo "DataQueue".
 **/
    private void showDqForm()
{
            // Creare il modulo per la coda dati.
            if (dataqueueForm_ == null)
    {
                    dataqueueForm_ = new Form(DATA_QUEUE);
                    dataqueueForm_.append(dqInputText_);
                    dataqueueForm_.append(dqOutputText_);
                    dataqueueForm_.append(dqReadOrWrite_);
                    dataqueueForm_.append(dqStatusText_);
                    dataqueueForm_.addCommand(actionBack_);
                    dataqueueForm_.addCommand(actionClear_);
                    dataqueueForm_.addCommand(actionGo_);
                    dataqueueForm_.setCommandListener(this);
                    dataqueueForm_.setTicker(ticker_);
    }

            display_.setCurrent(dataqueueForm_);
}


    private void feedback(String text, AlertType type)
{
            feedback(text, type, display_.getCurrent());
}

/**
 *  Questo metodo viene utilizzato per creare una finestra di dialogo e visualizzare le informazioni
 *  sul feedback utilizzando un Avviso per l'utente.
 **/
    private void feedback(String text, AlertType type, Displayable returnToForm)
{
```

```
                System.err.flush();
                System.out.flush();

        Alert alert = new Alert("Alert", text, null, type);

                if (type == AlertType.INFO)
                        alert.setTimeout(3000);  // millisecondi
                                else
                        alert.setTimeout(Alert.FOREVER);  // Richiedere all'utente di annullare l'avviso.

                display_.setCurrent(alert, returnToForm);
    }


        // Forzare una nuova stesura del modulo corrente.
        private void repaint()
    {
        Alert alert = new Alert("Updating display ...", null, null, AlertType.INFO);
                alert.setTimeout(1000);  // millisecondi

                display_.setCurrent(alert, display_.getCurrent());
    }



    /**
          * Time to pause, free any space we don't need right now.
          * Implementa il metodo astratto della classe Midlet.
     **/
        protected void pauseApp()
    {
                display_.setCurrent(null);
    }



    /**
          * Destroy must cleanup everything.
          * Implements abstract method of class Midlet.
     **/
        protected void destroyApp(boolean unconditional)
    {
                // Scollegarsi da iSeries se si sta eliminando Midlet o si sta uscendo.
                if (system_ != null)
        {
                                        try
            {
                                system_.disconnect();
            }
                                        catch (Exception e)
            {
            }
        }
    }
}
```

Esempi: classi Utility

Questa sezione elenca gli esempi di codice forniti in tutta la documentazione delle classi Utility di IBM Toolbox per Java.

AS/400ToolboxInstaller

- Esempio: utilizzo della classe AS400ToolboxInstaller
- Esempio: installazione di IBM Toolbox per Java utilizzando AS400ToolboxInstaller

- Esempio: installazione del pacchetto ACCESS dalla riga comandi
- Esempio: utilizzo della classe Graphical Toolbox dalla riga comandi

AS/400ToolboxJarMaker

- Esempio: estrazione di AS400.class e di tutte le relative classi dipendenti da jt400.jar
- Esempio: suddivisione di jt400.jar in una serie di file di 300KB
- Esempio: eliminazione di file inutilizzati da un file JAR
- Esempio: creazione di un file Jar più piccolo di 400KB omettendo le tabelle di conversione con il parametro -ccsid

CommandPrompter

- Esempio: utilizzo di CommandPrompter per richiedere ed eseguire un comando

Il seguente esonero di responsabilità si applica a tutti gli esempi IBM Toolbox per Java:

> **Esonero di responsabilità per gli esempi di codice**
>
> L'IBM fornisce una licenza non esclusiva per utilizzare tutti gli esempi del codice di programmazione da cui creare funzioni simili personalizzate, in base a richieste specifiche.
>
> Questo codice di esempio è fornito dall'IBM con la sola funzione illustrativa. Questi esempi non sono stati interamente testati in tutte le condizioni. IBM, perciò, non fornisce nessun tipo di garanzia o affidabilità implicita, rispetto alla funzionalità o alle funzioni di questi programmi.
>
> Tutti i programmi qui contenuti vengono forniti all'utente ″COSI' COME SONO″ senza garanzie di alcun tipo. Le garanzie implicite di non contraffazione, commerciabilità e adeguatezza a scopi specifici sono espressamente vietate.

Esempio: utilizzo di AS400ToolboxInstaller per installare e aggiornare IBM Toolbox per Java

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per informazioni legali importanti.

```
////////////////////////////////////////////////////////////////////////////
//
// Esempio di installazione/aggiornamento. Questo programma utilizza la classe AS400ToolboxInstaller
// per installare ed aggiornare il pacchetto IBM Toolbox per Java sulla stazione di lavoro.
//
// Il programma controlla il percorso di destinazione per il pacchetto IBM Toolbox per Java.
// Se il pacchetto non viene individuato, esso installa il pacchetto sulla stazione di lavoro.
// Se il pacchetto viene individuato, quest'ultimo verifica il percorso origine per gli aggiornamenti. Se
// vengono rilevati aggiornamenti questi vengono copiati sulla stazione di lavoro.
//
// Sintassi del comando:
//    checkToolbox source target
//
// Dove
//    source = ubicazione dei file sorgenti. Questo nome è nel formato URL.
//    target = ubicazione dei file di destinazione.
//
////////////////////////////////////////////////////////////////////////////

import java.io.*;
import java.util.*;
import java.net.*;
import utilities.*;

public class checkToolbox extends Object
{
     public static void main(String[] parameters)
```

```
{
         System.out.println( " " );



         // Continuare con l'installazione/aggiornamento solo se sono stati specificati i nomi
         // sia dell'origine che della destinazione.

      if (parameters.length >= 2)
      {

               // Il primo parametro rappresenta l'origine per i file, il secondo è la destinazione.

               String sourcePath = parameters[0];
               String targetPath = parameters[1];

               boolean installIt = false;
               boolean updateIt  = false;



               // E' stato creato un programma di lettura per richiamare l'immissione dall'utente.

         BufferedReader inputStream = new BufferedReader(new InputStreamReader(System.in),1);

                                  try
      {
         // Puntare al pacchetto origine. AS400ToolboxInstaller utilizza la classe
                     // URL per accedere ai file.

                     URL sourceURL  = new URL(sourcePath);



         // Verificare se il pacchetto è installato sul client. In caso contrario, chiedere all'utente
                     // se l'installazione deve essere effettuata in questa fase.

         if (AS400ToolboxInstaller.isInstalled("ACCESS", targetPath) == false)
         {
            System.out.print("IBM Toolbox for Java is not installed.  Install now (Y/N):");

                              String userInput = inputStream.readLine();

                              if ((userInput.charAt(0) == 'y')   ||
                                    (userInput.charAt(0) == 'Y'))
                                 installIt = true;
         }



         // Il pacchetto è installato. Controllare se i pacchetti devono essere copiati dal
                        // server.
Se la destinazione è priva di dati, richiedere all'utente se l'aggiornamento deve
                        // essere eseguito in questa fase.

                                 else
         {
            if (AS400ToolboxInstaller.isUpdateNeeded("ACCESS", targetPath, sourceURL) == true)
            {
               System.out.print("IBM Toolbox for Java is out of date.  Install fixes (Y/N):");

                                 String userInput = inputStream.readLine();

               if ((userInput.charAt(0) == 'y') || (userInput.charAt(0) == 'Y'))
                                    updateIt = true;
            }
                                    else
```

```
            System.out.println("Target directory is current, no update needed.");
      }



                  // Se il pacchetto deve essere installato o aggiornato.

                  if (updateIt || installIt)
      {
                        // Copiare i file dal server nella destinazione.

         AS400ToolboxInstaller.install("ACCESS", targetPath, sourceURL);




                        // Notificare l'esito positivo dell'installazione/aggiornamento.

                  System.out.println("");

                        if (installIt)
         System.out.println("Install successful!");
                              else
         System.out.println("Update Successful!");



                        // Indicare all'utente cosa deve essere aggiunto alla variabile d'ambiente
                        // CLASSPATH.

                        Vector classpathAdditions = AS400ToolboxInstaller.getClasspathAdditions();

         if (classpathAdditions.size() > 0)
         {
                     System.out.println("");
         System.out.println("Add the following to the CLASSPATH environment variable:");

                           for (int i = 0; i < classpathAdditions.size(); i++)
         {
                        System.out.print("   ");
                                    System.out.println((String)classpathAdditions.elementAt(i));
         }
         }



                        // Indicare all'utente cosa può essere eliminato dalla variabile d'ambiente
                        // CLASSPATH.

                        Vector classpathRemovals = AS400ToolboxInstaller.getClasspathRemovals();

         if (classpathRemovals.size() > 0)
         {
                     System.out.println("");
         System.out.println("Remove the following from the CLASSPATH environment variable:");

                           for (int i = 0; i < classpathRemovals.size(); i++)
         {
                        System.out.print("   ");
                                    System.out.println((String)classpathRemovals.elementAt(i));
         }
         }
      }
   }

                           catch (Exception e)
   {
```

```
            // Se una qualsiasi delle operazioni riportate sopra ha dato esito negativo
            // considerare non riuscita l'operazione ed emettere l'eccezione.

            System.out.println("Install/Update failed");
                    System.out.println(e);
         }
      }



            // Visualizzare il testo di aiuto quando i parametri non sono corretti.

                              else
      {
                  System.out.println("");
                  System.out.println("");
                  System.out.println("");
         System.out.println("Parameters are not correct.  Command syntax is:");
                  System.out.println("");
         System.out.println("  checkToolbox sourcePath targetPath");
                  System.out.println("");
                  System.out.println("Where");
                  System.out.println("");
         System.out.println("   sourcePath = source for IBM Toolbox for Java files");
         System.out.println("   targetPath = target for IBM Toolbox for Java files");
                  System.out.println("");
                  System.out.println("For example:");
                  System.out.println("");
         System.out.println("   checkToolbox http://mySystem/QIBM/ProdData/HTTP/Public/jt400/ d:\\jt400");
                  System.out.println("");
                  System.out.println("");
      }

               System.exit(0);
   }
}
```

Esempio: utilizzo di CommandPrompter

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
////////////////////////////////////////////////////////////////////////////////
//
// Esempio CommandPrompter. Questo programma utilizza CommandPrompter, CommandCall e
// AS400Message per richiedere un comando, eseguire il comando e visualizzare qualsiasi
// messaggio restituito in caso di mancata esecuzione del comando.
//
// Sintassi del comando:
// Prompter commandString
//
////////////////////////////////////////////////////////////////////////////////

import com.ibm.as400.ui.util.CommandPrompter;
import com.ibm.as400.access.AS400;
import com.ibm.as400.access.AS400Message;
import com.ibm.as400.access.CommandCall;
import javax.swing.JFrame;
import java.awt.FlowLayout;
public class Prompter
{
public static void main ( String args[] ) throws Exception
  {
    JFrame frame = new JFrame();
    frame.getContentPane().setLayout(new FlowLayout());
  AS400 system = new AS400("mySystem", "myUserId", "myPasswd");
    String cmdName = args[0];
```

```
    // Avviare il CommandPrompter
    CommandPrompter cp = new CommandPrompter(frame, system, cmdName);
    if (cp.showDialog() == CommandPrompter.OK)
    {
        String cmdString = cp.getCommandString();
    System.out.println("Command string: " + cmdString);

        // Eseguire il comando creato nel prompter.
        CommandCall cmd = new CommandCall(system, cmdString);
        if (!cmd.run())
      {
            AS400Message[] msgList = cmd.getMessageList();
            for (int i = 0; i < msgList.length; ++i)
        {
                          System.out.println(msgList[i].getText());
        }
      }
    }
  System.exit(0);
  }
}
```

Esempi: classi Vaccess

Questa sezione elenca gli esempi di codice forniti in tutta la documentazione delle classi Vaccess di IBM Toolbox per Java.

AS400Panes

- Esempio: creazione di un AS400DetailsPane per presentare l'elenco degli utenti definiti in systemAS400DetailsPane
- Esempio: caricamento del contenuto di un pannello di dettagli prima di aggiungerlo alla frame
- Esempio: utilizzo di AS400ListPane per presentare un elenco di utenti
- Esempio: utilizzo di AS400DetailsPane per visualizzare i messaggi restituiti da una chiamata al comando
- Esempio: utilizzo di AS400TreePane per visualizzare una vista ad albero di un indirizzario
- Esempio: utilizzo di AS400ExplorerPane per presentare varie risorse di stampa

Command call

- Esempio: creazione di CommandCallButton
- Esempio: aggiunta di ActionCompletedListener per elaborare tutti i messaggi iSeries che vengono creati da un comando
- Esempio: utilizzo di CommandCallMenuItem

Code dati

- Esempio: creazione di DataQueueDocument
- Esempio: utilizzo di DataQueueDocument

Eventi di errore

- Esempio: gestione degli eventi di errore
- Esempio: definizione di un listener di errori
- Esempio: utilizzo di un handler personalizzato per gestire eventi di errori

IFS (Integrated file system)

- Esempio: utilizzo di IFSFileDialog
- Esempio: utilizzo di IFSFileSystemView

- Esempio: utilizzo di IFSTextFileDocument

JDBC
- Esempio: utilizzo dell'unità di controllo JDBC per creare e popolare una tabella
- Esempio: utilizzo dell'unità di controllo JDBC per interrogare una tabella ed emettere il suo contenuto
- Esempio: creazione di AS400JDBCDataSourcePane

Lavori
- Esempio: creazione di un VJobList e presentazione dell'elenco in un AS400ExplorerPane
- Esempio: presentazione di un elenco di lavori in un pannello di ricerca

Messaggi
- Esempio: utilizzo di VMessageQueue

Chiamata al programma
- Esempio: creazione di un ProgramCallMenuItem
- Esempio: elaborazione di tutti i messaggi iSeries generati dal programma
- Esempio: aggiunta di due parametri
- Esempio: utilizzo di un ProgramCallButton in un'applicazione

Stampa
- Esempio: utilizzo di VPrinter
- Esempio: VPrinterOutput

Accesso al livello record
- Esempio: creazione di un oggetto RecordListTablePane per visualizzare tutti record inferiori o uguali ad una chiave
- Esempio: utilizzo di RecordListFormPane

SpooledFileViewer
- Esempio: creazione di un programma di visualizzazione del file di spool per visualizzare un file di spool creato precedentemente su iSeries

SQL
- Esempio: utilizzo di SQLQueryBuilderPane
- Esempio: utilizzo di SQLResultSetTablePane

SystemValues
- Esempio: creazione di una GUI del valore di sistema utilizzando il pannello AS400Explorer

Utenti e gruppi
- Esempio: creazione di un VUserList con AS400DetailsPane
- Esempio: utilizzo di un AS400ListPane per creare un elenco di utenti per la selezione

Il seguente esonero di responsabilità si applica a tutti gli esempi IBM Toolbox per Java:

Esonero di responsabilità per gli esempi di codice

L'IBM fornisce una licenza non esclusiva per utilizzare tutti gli esempi del codice di programmazione da cui creare funzioni simili personalizzate, in base a richieste specifiche.

Questo codice di esempio è fornito dall'IBM con la sola funzione illustrativa. Questi esempi non sono stati interamente testati in tutte le condizioni. IBM, perciò, non fornisce nessun tipo di garanzia o affidabilità implicita, rispetto alla funzionalità o alle funzioni di questi programmi.

Tutti i programmi qui contenuti vengono forniti all'utente ″COSI' COME SONO″ senza garanzie di alcun tipo. Le garanzie implicite di non contraffazione, commerciabilità e adeguatezza a scopi specifici sono espressamente vietate.

Esempio: utilizzo di VUserList

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
////////////////////////////////////////////////////////////////////////
//
// Esempio di VUserList.  Questo programma presenta un elenco di utenti su
// un sistema in un pannello elenco e consente la selezione di uno o più
// utenti.
//
// Sintassi del comando:
//    VUserListExample system
//
////////////////////////////////////////////////////////////////////////

import com.ibm.as400.access.*;
import com.ibm.as400.vaccess.*;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class VUserListExample
{


        private static AS400ListPane listPane;


        public static void main (String[] args)
    {
                // Se non è specificato un sistema, visualizzare il testo di aiuto e
                // uscire.
                if (args.length != 1)
        {
            System.out.println("Usage:  VUserListExample system");
                        return;
        }

                                    try
        {
                   // Creare un oggetto AS400.
Il nome di sistema viene passato
                        // come primo argomento della riga comandi.
                        AS400 system = new AS400 (args[0]);

            // Creare VUserList.  Rappresenta un elenco di utenti
                        // visualizzati nel pannello elenco.
                        VUserList userList = new VUserList (system);

                        // Creare una frame.
            JFrame f = new JFrame ("VUserList example");

            // Creare un adattatore finestra di dialogo errore.  Questo visualizza
                        // qualsiasi errore all'utente.
                        ErrorDialogAdapter errorHandler = new ErrorDialogAdapter (f);

                        // Creare un pannello elenco per visualizzare l'elenco utenti.
                        // Utilizzare load per richiamare le informazioni dal server.
```

```
                        listPane = new AS400ListPane (userList);
                        listPane.addErrorListener (errorHandler);
                        listPane.load ();

                        // Quando si chiude la frame, riportare gli utenti
                        // selezionati ed uscire.
                        f.addWindowListener (new WindowAdapter () {
                                public void windowClosing (WindowEvent event)
                {
                                        reportSelectedUsers ();
                          System.exit(0);
                }
            });

                        // Effettuare il layout della frame con il pannello elenco.
                    f.getContentPane ().setLayout (new BorderLayout ());
            f.getContentPane ().add ("Center", listPane);
                    f.pack ();
                    f.show ();
        }
                                    catch (Exception e)
        {
                      System.out.println ("Error: " + e.getMessage ());
                 System.exit(0);
        }
    }


        private static void reportSelectedUsers ()
    {
                VObject[] selectedUsers = listPane.getSelectedObjects ();

                if (selectedUsers.length == 0)
            System.out.println ("No users were selected.");
                            else
        {
            System.out.println ("The selected users were:");
                        for (int i = 0; i < selectedUsers.length; ++i)
                                System.out.println (selectedUsers[i]);
        }
    }


}
```

Esempio: utilizzo di VMessageList

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
///////////////////////////////////////////////////////////////////////////
//
// Esempio di VMessageList.  Questo programma presenta una vista
// dettagliata di messaggi restituiti da una chiamata al comando.
//
// Sintassi del comando:
//    VMessageListExample system
//
// Questo sorgente è un esempio di "VMessageList" di IBM Toolbox per Java.
//
///////////////////////////////////////////////////////////////////////////

import com.ibm.as400.access.*;
import com.ibm.as400.vaccess.*;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
```

```
public class VMessageListExample
{


        public static void main (String[] args)
    {
            // Se non è stato specificato un sistema, visualizzare il testo di aiuto ed
            // uscire.
            if (args.length != 1)
        {
            System.out.println("Usage:  VMessageListExample system");
                        return;
        }

                                    try
        {
                  // Creare un oggetto AS400.
Il nome di sistema è stato passato
                        // come primo argomento della riga comandi.
                        AS400 system = new AS400 (args[0]);

                        // Creare un oggetto CommandCall per eseguire il comando.
                        CommandCall command = new CommandCall (system);
            command.run ("CRTLIB FRED");

                        // Creare un oggetto VMessageList con i messaggi
                        // restituiti da una chiamata al comando.
                        VMessageList messageList = new VMessageList (command.getMessageList ());

                        // Creare una frame.
            JFrame f = new JFrame ("VMessageList example");

            // Creare un adattatore finestra di dialogo errore.  Questo visualizzerà
                        // qualsiasi errore all'utente.
                        ErrorDialogAdapter errorHandler = new ErrorDialogAdapter (f);

                        // Creare un pannello di dettagli per visualizzare l'elenco di messaggi.
                        // Utilizzare load per caricare le informazioni.
                        AS400DetailsPane detailsPane = new AS400DetailsPane (messageList);
                        detailsPane.addErrorListener (errorHandler);
                        detailsPane.load ();

                        // Quando si chiude la frame, uscire.
                        f.addWindowListener (new WindowAdapter () {
                                public void windowClosing (WindowEvent event)
                {
                          System.exit(0);
                }
            });

                        // Effettuare il layout della frame con il pannello dettagli.
                    f.getContentPane ().setLayout (new BorderLayout ());
                        f.getContentPane ().add ("Center", detailsPane);
                    f.pack ();
                    f.show ();
        }
                                    catch (Exception e)
        {
                      System.out.println ("Error: " + e.getMessage ());
                 System.exit(0);
        }
    }


}
```

Esempio: utilizzo di VIFSDirectory

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
///////////////////////////////////////////////////////////////////////
//
// Esempio di VIFSDirectory.  Questo programma presenta una vista ad albero
// di alcuni indirizzari nell'IFS.
//
// Sintassi del comando:
//    VIFSDirectoryExample system
//
// Questo sorgente è un esempio di "VIFSDirectory" di IBM Toolbox per Java.
//
///////////////////////////////////////////////////////////////////////

import com.ibm.as400.access.*;
import com.ibm.as400.vaccess.*;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class VIFSDirectoryExample
{


        public static void main (String[] args)
    {
                // Se non è stato specificato un sistema, visualizzare il testo di aiuto ed
                // uscire.
                if (args.length != 1)
        {
            System.out.println("Usage:  VIFSDirectoryExample system");
                        return;
        }

                                    try
        {
                  // Creare un oggetto AS400.
Il nome di sistema è stato passato
                        // come primo argomento della riga comandi.
                        AS400 system = new AS400 (args[0]);

                        // Creare un oggetto VIFSDirectory che rappresenti la root
                        // dell'albero indirizzari che si sta per visualizzare.
            VIFSDirectory directory = new VIFSDirectory (system, "/QIBM/ProdData");

                        // Creare una frame.
            JFrame f = new JFrame ("VIFSDirectory example");

            // Creare un adattatore finestra di dialogo errore.  Questo visualizzerà
                        // qualsiasi errore all'utente.
                        ErrorDialogAdapter errorHandler = new ErrorDialogAdapter (f);

                        // Creare un pannello albero per presentare una gerarchia degli indirizzari
                        // Caricare le informazioni dal sistema.
                        AS400TreePane treePane = new AS400TreePane (directory);
                        treePane.addErrorListener (errorHandler);
                        treePane.load ();

                        // Quando si chiude la frame, uscire.
                        f.addWindowListener (new WindowAdapter () {
                                public void windowClosing (WindowEvent event)
                {
                                  System.exit(0);
                }
            });
```

```
                        // Effettuare il layout della frame con il pannello albero.
                    f.getContentPane ().setLayout (new BorderLayout ());
            f.getContentPane ().add ("Center", treePane);
                    f.pack ();
                    f.show ();
        }
                                    catch (Exception e)
        {
                      System.out.println ("Error: " + e.getMessage ());
                  System.exit(0);
        }
    }


}
```

Esempio: utilizzo di VPrinters

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
////////////////////////////////////////////////////////////////////////
//
// Esempio VPrinters.  Questo programma presenta varie risorse
// stampa di rete con un pannello explorer.
//
// Sintassi del comando:
//    VPrintersExample system
//
// Questo sorgente è un esempio "VPrinters" IBM Toolbox per Java .
//
////////////////////////////////////////////////////////////////////////

import com.ibm.as400.access.*;
import com.ibm.as400.vaccess.*;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class VPrintersExample
{


        public static void main (String[] args)
    {
                // Se non è stato specificato un sistema, visualizzare il testo di aiuto ed
                // uscire.
                if (args.length != 1)
        {
            System.out.println("Usage:  VPrintersExample system");
                        return;
        }

                                    try
        {
                     // Creare un oggetto AS400.
Il nome di sistema è stato passato
                        // come primo argomento della riga comandi.
                        AS400 system = new AS400 (args[0]);

                        // Creare un oggetto VPrinters che rappresenti l'elenco
                        // di stampanti collegate al sistema.
                        VPrinters printers = new VPrinters (system);

                        // Creare una frame.
            JFrame f = new JFrame ("VPrinters example");
```

```
            // Creare un adattatore finestra di dialogo errore.  Questo visualizzerà
                        // qualsiasi errore all'utente.
                        ErrorDialogAdapter errorHandler = new ErrorDialogAdapter (f);

                        // Creare un pannello explorer per presentare le risorse stampa di rete.
                     // Utilizzare load per caricare le informazioni dal sistema.
                        AS400ExplorerPane explorerPane = new AS400ExplorerPane (printers);
                        explorerPane.addErrorListener (errorHandler);
                          explorerPane.load ();

                        // Quando si chiude la frame, uscire.
                        f.addWindowListener (new WindowAdapter () {
                                public void windowClosing (WindowEvent event)
                    {
                                  System.exit(0);
                    }
                });

                        // Effettuare il layout della frame con il pannello explorer.
                       f.getContentPane ().setLayout (new BorderLayout ());
                        f.getContentPane ().add ("Center", explorerPane);
                       f.pack ();
                       f.show ();
            }
                                        catch (Exception e)
            {
                         System.out.println ("Error: " + e.getMessage ());
                     System.exit(0);
            }
        }


}
```

Esempio: utilizzo di CommandCallMenuItem

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per informazioni legali importanti.

```
///////////////////////////////////////////////////////////////////////////
//
// Esempio della voce di menu CommandCall. Questo programma mostra come
// utilizzare una voce di menu che richiama un comando server. Visualizzerà
// qualsiasi messaggio riportato in una finestra di dialogo.
//
// Sintassi del comando:
//    CommandCallMenuItemExample system
//
///////////////////////////////////////////////////////////////////////////

import com.ibm.as400.access.*;
import com.ibm.as400.vaccess.*;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class CommandCallMenuItemExample
{



        private static JFrame f;



        public static void main (String[] args)
    {
                // Se non è stato specificato un sistema, visualizzare il testo di aiuto ed
```

```
            // uscire.
            if (args.length != 1)
        {
            System.out.println("Usage:  CommandCallMenuItemExample system");
                        return;
        }

                                    try
        {
            // Creare un oggetto AS400.
Il nome di sistema è passato
                        // come primo argomento della riga comandi.
                        AS400 system = new AS400 (args[0]);

                        // Creare una frame.
            f = new JFrame ("Command call menu item example"

            // Creare un adattatore di finestra di dialogo di errore. Quest'ultimo mostrerà
                        // qualsiasi errore all'utente.
                        ErrorDialogAdapter errorHandler = new ErrorDialogAdapter (f);

                        // Creare un oggetto CommandCallMenuItem per eseguire il comando.
            CommandCallMenuItem menuItem =
               new CommandCallMenuItem ("Clear library FRED", null, system, "CLRLIB FRED");
                        menuItem.addErrorListener (errorHandler);

                        // Aggiungere un listener completato dell'azione per visualizzare qualsiasi
                        // messaggio restituito in una finestra di dialogo.
                        menuItem.addActionCompletedListener (new ActionCompletedListener ()
            {
                                public void actionCompleted (ActionCompletedEvent event)
                {
                    // Richiamare l'elenco dei messaggi dall'origine evento.
                    CommandCallMenuItem item = (CommandCallMenuItem) event.getSource ();
                    AS400Message[] messageList = item.getMessageList ();

                    // Utilizzare un AS400DetailsPane per visualizzare i messaggi.
                    VMessageList vmessageList = new VMessageList (messageList);
                    AS400DetailsPane messageDetails = new AS400DetailsPane (vmessageList);
                    messageDetails.load ();

                    // Visualizzare i dettagli in una finestra di dialogo.
                    JDialog dialog = new JDialog(f);
                    dialog.getContentPane().setLayout(new BorderLayout());
                    dialog.getContentPane().add("Center"messageDetails);
                    dialog.pack();
                    dialog.setVisible(true);
                }
            });

            // Creare un menu contenente la voce.
            JMenu menu = new JMenu ("Server Command Calls");
            menu.add (menuItem);

            JMenuBar menuBar = new JMenuBar ();
            menuBar.add (menu);

            f.getRootPane ().setJMenuBar (menuBar);

            // Quando si chiude la frame, uscire.
            f.addWindowListener (new WindowAdapter () {
                public void windowClosing (WindowEvent event)
                {
                    System.exit (0);
                }
            });
```

```
            // Effettuare il layout della frame con il pannello dettagli.
            f.getContentPane ().setLayout (new BorderLayout ());
                        f.setSize (300, 400);
                    f.show ();
        }
                                    catch (Exception e)
        {
                     System.out.println ("Error: " + e.getMessage ());
           System.exit (0);
        }
    }


}
```

Esempio: utilizzo di DataQueueDocument

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per informazioni legali importanti.

```
////////////////////////////////////////////////////////////////////////
//
// Esempio di documento coda dati. Questo programma mostra come
// utilizzare un documento associato ad una coda dati server.
//
// Sintassi del comando:
//    DataQueueDocumentExample system read|write
//
////////////////////////////////////////////////////////////////////////

import com.ibm.as400.access.*;
import com.ibm.as400.vaccess.*;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class DataQueueDocumentExample
{
        private static DataQueueDocument    dqDocument;
        private static JTextField           text;
        private static boolean              rw;

        public static void main (String[] args)
    {
                // Se non è stato specificato un sistema o read|write, visualizzare
        // il testo di aiuto ed uscire.
                if (args.length != 2)
        {
            System.out.println("Usage:  DataQueueDocumentExample system read|write");
                        return;
        }

        rw = args[1].equalsIgnoreCase ("read");
        String mode = rw ? "Read" : "Write";

                                    try
        {
                        // Creare due frame.
            JFrame f =
               new JFrame ("Data queue document example - " + mode);

            // Creare un adattatore di finestra di dialogo di errore. Quest'ultimo mostrerà
                        // qualsiasi errore all'utente.
                        ErrorDialogAdapter errorHandler = new ErrorDialogAdapter (f);

            // Creare un adattatore del cursore di lavoro. Questa operazione adatterà
                        // il cursore ogni volta che una coda dati viene scritta o letta.
                        WorkingCursorAdapter cursorAdapter = new WorkingCursorAdapter (f);
```

```
            // Creare un oggetto AS400.
Il nome di sistema è passato
                        // come primo argomento della riga comandi.
                        AS400 system = new AS400 (args[0]);

                        // Creare il nome percorso coda dati.
            QSYSObjectPathName dqName = new QSYSObjectPathName ("QGPL", "JAVATALK", "DTAQ");

                        // Assicurarsi che esista la coda dati.
                        DataQueue dq = new DataQueue (system, dqName.getPath ());
                                        try
            {
                                dq.create (200);
            }
                                        catch (Exception e)
            {
                // Ignorare eccezioni. Molto probabilmente, la coda dati
                                // esiste già.
            }

                        // Creare un oggetto DataQueueDocument.
                        dqDocument = new DataQueueDocument (system, dqName.getPath ());
                        dqDocument.addErrorListener (errorHandler);
                        dqDocument.addWorkingListener (cursorAdapter);

                        // Creare un campo di testo utilizzato per presentare il documento.
            text = new JTextField (dqDocument, "", 40);
            text.setEditable (! rw);

                        // Durante l'esecuzione del programma, è necessario poter controllare quando
            // si verifica l'operazione di lettura e di scrittura. Sarà possibile
                        // effettuare tale controllo tramite un pulsante.
                        Button button = new Button (mode);
                        button.addActionListener (new ActionListener ()
            {
                                public void actionPerformed (ActionEvent event)
                {
                                        if (rw)
                                                dqDocument.read ();
                          else {
                                                dqDocument.write ();
                        text.setText ("");
                    }
                }
            });

                        // Quando la frame si chiude, uscire dal programma.
                        f.addWindowListener (new WindowAdapter () {
                                public void windowClosing (WindowEvent event)
                {
                          System.exit(0);
                }
            });

                        // Effettuare il layout della frame.
                        f.getContentPane ().setLayout (new FlowLayout ());
                        f.getContentPane ().add (text);
                        f.getContentPane ().add (button);
                    f.pack ();
                    f.show ();
        }
                                    catch (Exception e)
        {
                                System.out.println ("Error: " + e.getMessage ());
```

```
                System.exit(0);
        }
    }
}
```

Esempio: utilizzo di IFSFileDialog

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per informazioni legali importanti.

```
////////////////////////////////////////////////////////////////////////////
//
// Esempio di File Dialog.
//
////////////////////////////////////////////////////////////////////////////

import java.io.*;
import java.awt.*;
import com.ibm.as400.access.*;
import com.ibm.as400.vaccess.*;

public class FileDialogExample extends Object
{

     public static void main(String[] parameters)
   {

            System.out.println( " " );



            // se non è stato specificato un nome di sistema, visualizzare il testo di aiuto ed uscire.

      if (parameters.length >= 1)
      {

                  // Il primo parametro è il sistema che contiene i file.

                  String system = parameters[0];



                                    try
         {
                        // Creare un oggetto AS400 per il server che contiene i file.
             // Collegarsi al server del file sul server. Collegarsi ora in modo tale che
                        // lo schermo di collegamento venga visualizzato.

                       AS400 as400 = new AS400(system);
                        as400.connectService(AS400.FILE);



                        // Creare una frame per contenere la finestra di dialogo.

                        Frame frame = new Frame();



                        // Creare l'oggetto finestra di dialogo file.

             IFSFileDialog fileDialog = new IFSFileDialog(frame, "File Open", as400);



             // Creare l'elenco dei filtri che l'utente può scegliere quindi aggiungere i filtri
             // alla finestra di dialogo.
```

```
        FileFilter[] filterList = {
           new FileFilter("All files (*.*)", "*.*"),
           new FileFilter("Executables (*.exe)", "*.exe"),
           new FileFilter("HTML files (*.html)", "*.html"),
           new FileFilter("Images (*.gif)", "*.gif"),
           new FileFilter("Text files (*.txt)", "*.txt")};

                    fileDialog.setFileFilter(filterList, 0);



        // Impostare il testo per il pulsante "OK" sulla finestra di dialogo.

        fileDialog.setOkButtonText("Open");



        // Impostare il testo per il pulsante "Annulla" sulla finestra di dialogo.

        fileDialog.setCancelButtonText("Cancel");



                    // Impostare l'indirizzario iniziale per la finestra di dialogo.

        fileDialog.setDirectory("/");



        // Visualizzare la finestra di dialogo ed attendere che l'utente prema OK o Annulla

                    int pressed = fileDialog.showDialog();



        // Se l'utente ha premuto OK, richiamare il percorso completo ed il nome
        // del file selezionato.

                    if (pressed == IFSFileDialog.OK)
        {
           System.out.println("User selected: " +
                              fileDialog.getAbsolutePath());
        }



                    // Invece se l'utente ha premuto Annulla, visualizzare un messaggio.

                    else if (pressed == IFSFileDialog.CANCEL)
        {
           System.out.println("User pressed cancel");
        }

                                 else
           System.out.println("User didn't press Open or Cancel");

     }
       catch(Exception e)
     {
        // Se una qualsiasi delle operazioni riportate sopra ha dato esito negativo considerare
        // non riuscita l'operazione della finestra di dialogo ed emettere l'eccezione.

        System.out.println("Dialog operation failed");
                System.out.println(e);
     }
}
```

```
      // Visualizzare il testo di aiuto quando i parametri non sono corretti.

                              else
      {
                  System.out.println("");
                  System.out.println("");
                  System.out.println("");
         System.out.println("Parameters are not correct.  Command syntax is:");
                  System.out.println("");
         System.out.println("  FileDialogExample system");
                  System.out.println("");
                  System.out.println("Where");
                  System.out.println("");
         System.out.println("  system = iSeries server");
                  System.out.println("");
                  System.out.println("For example:");
                  System.out.println("");
                  System.out.println("");
         System.out.println("  FileDialogExample mySystem");
                  System.out.println("");
                  System.out.println("");
      }

      System.exit(0);
   }
}
```

Esempio: utilizzo di IFSTextFileDocument

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per informazioni legali importanti.

```
///////////////////////////////////////////////////////////////////////////
//
// Esempio di documento file di testo IFS. tale programma mostra come
// utilizzare un documento associato ad un file di testo nell'Integrated File
// System di AS/400.
//
// Sintassi del comando:
//    IFSTextFileDocumentExample system path
//
///////////////////////////////////////////////////////////////////////////

import com.ibm.as400.access.*;
import com.ibm.as400.vaccess.*;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class IFSTextFileDocumentExample
{



        private static IFSTextFileDocument  document;
        private static JTextPane            text;



        public static void main (String[] args)
    {
                // Se non è stato specificato un sistema o un percorso, visualizzare
        // il testo di aiuto ed uscire.
                if (args.length != 2)
        {
            System.out.println("Usage:  IFSTextFileDocumentExample system path");
                        return;
```

```
        }

                                    try
        {
                        // Creare due frame.
            JFrame f = new JFrame ("IFS text file document example");

            // Creare un adattatore di finestra di dialogo di errore. Quest'ultimo mostrerà
                        // qualsiasi errore all'utente.
                        ErrorDialogAdapter errorHandler = new ErrorDialogAdapter (f);

            // Creare un adattatore del cursore di lavoro. Questa operazione adatterà
                        // il cursore ogni volta che il file di testo viene letto o scritto.
                        WorkingCursorAdapter cursorAdapter = new WorkingCursorAdapter (f);

                                    // Creare un oggetto AS400.
Il nome di sistema è passato
                        // come primo argomento della riga comandi.
                        AS400 system = new AS400 (args[0]);

                        // Creare e caricare il documento file di testo IFS.
                        document = new IFSTextFileDocument (system, args[1]);
                        document.addErrorListener (errorHandler);
                        document.addWorkingListener (cursorAdapter);
                                document.load ();

                        // Creare il pannello di testo utilizzato per presentare il documento.
                        text = new JTextPane (document);
                        text.setSize (new Dimension (500, 500));

                        // Impostare un pannello a scorrimento da utilizzare con il pannello di testo.
                        JScrollPane scroll = new JScrollPane (text);
                        scroll.setHorizontalScrollBarPolicy (JScrollPane.HORIZONTAL_SCROLLBAR_AS_NEEDED);
                        scroll.setVerticalScrollBarPolicy (JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED);

                        // Creare una barra di menu con un singolo menu.
                        MenuBar menuBar = new MenuBar ();
            Menu menu = new Menu ("File");
                        menuBar.add (menu);

                        // Aggiungere le voci di menu da caricare e salvare.
            MenuItem load = new MenuItem ("Load");
                        load.addActionListener (new ActionListener ()
            {
                                public void actionPerformed (ActionEvent event)
                {
                                        document.load ();
                }
            });
                        menu.add (load);

            MenuItem save = new MenuItem ("Save");
                        save.addActionListener (new ActionListener ()
            {
                                public void actionPerformed (ActionEvent event)
                {
                                        document.save ();
                }
            });
                        menu.add (save);

                        // Quando la frame si chiude, uscire dal programma.
                        f.addWindowListener (new WindowAdapter () {
                                public void windowClosing (WindowEvent event)
                {
                    System.exit (0);
                }
```

```
            });

                        // Effettuare il layout della frame.
            f.getContentPane ().setLayout (new BorderLayout ());
            f.getContentPane ().add ("Center", scroll);
                        f.setMenuBar (menuBar);
                    f.pack ();
                    f.show ();
        }
                                    catch (Exception e)
        {
                              System.out.println ("Error: " + e.getMessage ());
           System.exit (0);
        }
    }


}
```

JDBCDataSourcePane di AS400

La classe AS400JDBCDataSourcePane mostra i valori proprietà di un oggetto AS400JDBCDataSource.Facoltativamente, è possibile effettuare delle modifiche all'oggetto AS400JDBCDataSource.

AS400JDBCDataSourcePane estende JComponent. Per utilizzare AS400JDBCDataSourcePane per visualizzare le proprietà di un sorgente dati, questo può essere specificata nel programma di creazione AS400JDBCDataSourcePane o impostato dopo che AS400JDBCDataSourcePane è stato creato utilizzando setDataSource(). Le modifiche effettuate alla GUI (graphical user interface) possono essere applicate all'oggetto sorgente dati utilizzando applyChanges().

Esempio: utilizzo di AS400JDBCDataSourcePane

Il seguente esempio crea un AS400JDBCDataSourcePane e un pulsante **OK** e li aggiunge ad una frame. Le modifiche effettuate alla GUI vengono applicate al sorgente dati facendo clic su **OK**.

```
                // Creare un'origine dati.
                myDataSource = new AS400JDBCDataSource();

                // Creare una finestra per contenere il pannello e un pulsante OK.
                JFrame frame = new JFrame ("JDBC Data Source Properties");

                // Creare un pannello origine dati.
                dataSourcePane = new AS400JDBCDataSourcePane(myDataSource);

                // Creare un pulsante OK
                JButton okButton = new JButton("OK");

        // Aggiungere un ActionListener al pulsante OK. Quando il pulsante OK
                // viene premuto, applyChanges() verrà richiamato per sincronizzare
                // le modifiche all'origine dati.
                okButton.addActionListener(new ActionListener()
           {
                            public void actionPerformed(ActionEvent ev)
              {
                                  // Applicare tutte le modifiche effettuate sul pannello origine dati
                // all'origine dati. Se tutte le modifiche vengono applicate con
                                  // esito positivo, richiamare l'origine dati dal pannello.
                                  if (dataSourcePane.applyChanges())
                {
                                        System.out.println("ok pressed");
                                        myDataSource = dataSourcePane.getDataSource();
                                        System.out.println(myDataSource.getServerName());
                }
              }
```

```
            }
        );

                // Impostare la frame per visualizzare il pannello e il pulsante OK.
                frame.getContentPane ().setLayout (new BorderLayout ());
                frame.getContentPane ().add ("Center", dataSourcePane);
                frame.getContentPane ().add ("South", okButton);

                // Comprimere la frame.
                frame.pack ();

                //Visualizzare il pannello e il pulsante OK.
        frame.show();
```

Esempio: utilizzo di VJobList per presentare un elenco di lavori

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
////////////////////////////////////////////////////////////////////////
//
// Esempio di elenco lavori.  Questo programma presenta un elenco di lavori in un
// pannello explorer.
//
// Sintassi del comando:
//    VJobListExample system
//
// Questo sorgente è un esempio di "AS400ExplorerPane" di IBM Toolbox per Java.
//
////////////////////////////////////////////////////////////////////////

import com.ibm.as400.access.*;
import com.ibm.as400.vaccess.*;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class VJobListExample
{


        public static void main (String[] args)
    {
                // Se non è stato specificato un sistema, visualizzare il testo di aiuto ed
                // uscire.
                if (args.length != 1)
        {
            System.out.println("Usage:  VJobListExample system");
                        return;
        }

                                    try
        {
                    // Creare un oggetto AS400.
Il nome di sistema è stato passato
                        // come primo argomento della riga comandi.
                        AS400 system = new AS400 (args[0]);

                        // Creare un oggetto VJobList che rappresenti l'elenco
                        // di lavori denominata QZDASOINIT.
                        VJobList jobList = new VJobList (system);
            jobList.setName ("QZDASOINIT");

                        // Creare una frame.
            JFrame f = new JFrame ("Job list example");

            // Creare un adattatore finestra di dialogo errore.  Questo visualizzerà
                        // qualsiasi errore all'utente.
```

```
                        ErrorDialogAdapter errorHandler = new ErrorDialogAdapter (f);

                        // Creare un pannello explorer per presentare l'elenco di lavori.
                 // Utilizzare load per caricare le informazioni dal sistema.
                        AS400ExplorerPane explorerPane = new AS400ExplorerPane (jobList);
                        explorerPane.addErrorListener (errorHandler);
                        explorerPane.load ();

                        // Quando si chiude la frame, uscire.
                        f.addWindowListener (new WindowAdapter () {
                                public void windowClosing (WindowEvent event)
                {
                          System.exit(0);
                }
            });

                        // Effettuare il layout della frame con il pannello explorer.
                    f.getContentPane ().setLayout (new BorderLayout ());
                        f.getContentPane ().add ("Center", explorerPane);
                    f.pack ();
                    f.show ();
        }
                                    catch (Exception e)
        {
                      System.out.println ("Error: " + e.getMessage ());
                 System.exit(0);
        }
    }


}
```

Esempio: utilizzo di VMessageQueue

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
///////////////////////////////////////////////////////////////////////////
//
// Esempio di coda messaggi.  Questo programma presenta una coda messaggi in un
// pannello explorer.
//
// Sintassi del comando:
//    VMessageQueueExample system
//
// Questo sorgente è un esempio di "VMessageQueue" di IBM Toolbox per Java.
//
///////////////////////////////////////////////////////////////////////////

import com.ibm.as400.access.*;
import com.ibm.as400.vaccess.*;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class VMessageQueueExample
{


        public static void main (String[] args)
    {
                // Se non è stato specificato un sistema, visualizzare il testo di aiuto ed
                // uscire.
                if (args.length != 1)
        {
            System.out.println("Usage:  VMessageQueueExample system");
                        return;
        }
```

```
                                try
        {
                   // Creare un oggetto AS400.
Il nome di sistema è stato passato
                        // come primo argomento della riga comandi.
                        AS400 system = new AS400 (args[0]);

                        // Forzare l'utente al collegamento in modo da saperne l'id utente.
                        system.connectService (AS400.COMMAND);

                        // Creare un oggetto VMessageQueue che rappresenti la
                        // coda messaggi dell'utente corrente.
                        VMessageQueue queue = new VMessageQueue (system,
                    QSYSObjectPathName.toPath ("QUSRSYS", system.getUserId (),
                    "MSGQ"));

                        // Creare una frame.
            JFrame f = new JFrame ("Message queue example");

            // Creare un adattatore finestra di dialogo errore.  Questo visualizzerà
                        // qualsiasi errore all'utente.
                        ErrorDialogAdapter errorHandler = new ErrorDialogAdapter (f);

                        // Creare un pannello explorer per presentare la coda messaggi.
                 // Utilizzare load per caricare le informazioni dal sistema.
                        AS400ExplorerPane explorerPane = new AS400ExplorerPane (queue);
                        explorerPane.addErrorListener (errorHandler);
                        explorerPane.load ();

                        // Quando si chiude la frame, uscire.
                        f.addWindowListener (new WindowAdapter () {
                                public void windowClosing (WindowEvent event)
                    {
                              System.exit(0);
                    }
            });

                        // Effettuare il layout della frame con il pannello explorer.
                    f.getContentPane ().setLayout (new BorderLayout ());
                        f.getContentPane ().add ("Center", explorerPane);
                    f.pack ();
                    f.show ();
        }
                                    catch (Exception e)
        {
                      System.out.println ("Error: " + e.getMessage ());
                 System.exit(0);
        }
    }


}
```

Esempio: utilizzo di un pulsante per chiamare un programma sul server

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
///////////////////////////////////////////////////////////////////////////
//
// Esempio di pulsante per la chiamata ad un programma.  Questo programma illustra come
// utilizzare un pulsante che chiama un programma sul server.  Esso scambierà dati
// con il programma server attraverso un parametro di immissione ed emissione.
//
// Sintassi del comando:
//    ProgramCallButtonExample system
//
```

```
// Questo sorgente è un esempio di "ProgramCallButton" di IBM Toolbox per Java.
//
///////////////////////////////////////////////////////////////////////////

import com.ibm.as400.access.*;
import com.ibm.as400.vaccess.*;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class ProgramCallButtonExample
{

        private ProgramParameter   parm1, parm2, parm3, parm4, parm5;
        private JTextField         cpuField;
        private JTextField         dasdField;
        private JTextField         jobsField;


        // Creare un oggetto ProgramCallButtonExample, quindi chiamare la
    // versione non statica di main().  Se non si fa questo
    // le variabili di classe (parm1, parm2, ...) devono essere dichiarate
    // statiche.  Se sono statiche non possono essere utilizzate dal
        // listener completo dell'azione in Java 1.1.7 o 1.1.8.
        public static void main (String[] args)
    {
              ProgramCallButtonExample me = new ProgramCallButtonExample();
              me.Main(args);
    }

        public void Main (String[] args)
    {
                // Se non è stato specificato un sistema, visualizzare il testo di aiuto ed
                // uscire.
                if (args.length != 1)
        {
            System.out.println("Usage:  ProgramCallButtonExample system");
                        return;
        }

                                    try
        {
                        // Creare una frame.
            JFrame f = new JFrame ("Program call button example");

            // Creare un adattatore finestra di dialogo errore.  Questo visualizzerà
                        // qualsiasi errore all'utente.
                        ErrorDialogAdapter errorHandler = new ErrorDialogAdapter (f);

                                  // Creare un oggetto AS400.
Il nome di sistema è stato passato
                        // come primo argomento della riga comandi.
                        AS400 system = new AS400 (args[0]);

                        // Creare il nome percorso programma.
            QSYSObjectPathName programName = new QSYSObjectPathName ("QSYS",
                "QWCRSSTS", "PGM");

            // Creare un oggetto ProgramCallButton.  Il pulsante
            // conterrà il testo "Aggiorna" e nessuna icona.
            ProgramCallButton button = new ProgramCallButton ("Refresh", null);
                        button.setSystem (system);
                        button.setProgram (programName.getPath ());
                        button.addErrorListener (errorHandler);

                        // Il primo parametro è un parametro di emissione a 64 byte.
                        parm1 = new ProgramParameter (64);
```

```
            button.addParameter (parm1);

            // Si utilizza il secondo parametro per impostare la dimensione buffer
// del primo parametro.  Verrà sempre impostata su
// 64.  Si tenga a mente che occorre convertire il valore int
            // Java 64 nel formato utilizzato nel server.
            AS400Bin4 parm2Converter = new AS400Bin4 ();
            byte[] parm2Bytes = parm2Converter.toBytes (64);
            parm2 = new ProgramParameter (parm2Bytes);
            button.addParameter (parm2);

// Il terzo parametro è il formato dello stato.  Si
// utilizzerà sempre "SSTS0200".  Questo è un valore String e
            // di nuovo è necessario convertirlo nel formato utilizzato nel server.
            AS400Text parm3Converter = new AS400Text (8, system);
byte[] parm3Bytes = parm3Converter.toBytes ("SSTS0200");
            parm3 = new ProgramParameter (parm3Bytes);
            button.addParameter (parm3);

            // Il quarto parametro è il parametro delle statistiche di reimpostazione.
// Verrà sempre passato "*NO" come String a 10 caratteri.
            AS400Text parm4Converter = new AS400Text (10, system);
byte[] parm4Bytes = parm4Converter.toBytes ("*NO       ");
            parm4 = new ProgramParameter (parm4Bytes);
            button.addParameter (parm4);

// Il quinto parametro è relativo alle informazioni sull'errore.  E'
// un parametro di immissione/emissione.  Non verrà utilizzato
            // per questo esempio, ma è necessario impostarlo su qualche valore
            // oppure il numero dei parametri non corrisponderà
            // a quello previsto dal server.
            byte[] parm5Bytes = new byte[32];
            parm5 = new ProgramParameter (parm5Bytes, 0);
            button.addParameter (parm5);

            // Quando si esegue il programma, si otterrà un gruppo di dati.
            // E' necessario fare in modo che l'utente li visualizzi.
            // In questo caso, verranno utilizzati semplici etichette e campi
            // di testo.
JLabel cpuLabel = new JLabel ("CPU Utilitization: ");
            cpuField = new JTextField (10);
            cpuField.setEditable (false);

JLabel dasdLabel = new JLabel ("DASD Utilitization: ");
            dasdField = new JTextField (10);
            dasdField.setEditable (false);

JLabel jobsLabel = new JLabel ("Number of active jobs: ");
            jobsField = new JTextField (10);
            jobsField.setEditable (false);

            // Quando si chiude la frame, uscire.
            f.addWindowListener (new WindowAdapter ()
{
                    public void windowClosing (WindowEvent event)
    {
              System.exit(0);
    }
});

            // Quando il programma viene chiamato, è necessario elaborare le
            // informazioni restituite nel primo parametro.
            // Il formato dei dati in questo parametro è stato documentato
            // dal programma che si sta chiamando.
            button.addActionCompletedListener (new ActionCompletedListener ()
{
                    public void actionCompleted (ActionCompletedEvent event)
```

```
            {
                                        try
                {
                                       // Richiamare i dati dal primo parametro.
                                       // Sono nel formato server.
                                       byte[] parm1Bytes = parm1.getOutputData ();

                                       // Ogni porzione di dati necessaria
                   // è un int.  E' possibile creare un programma di conversione
                                       // per effettuare tutte le conversioni.
                                       AS400Bin4 parm1Converter = new AS400Bin4 ();

                                       // Richiamare l'utilizzo CPU che inizia al byte 32.
                                       // Impostare questo valore nel campo testo corrispondente.
                                       int cpu = parm1Converter.toInt (parm1Bytes, 32);
                   cpuField.setText (Integer.toString (cpu / 10) + "%");

                                       // Richiamare l'utilizzo DASD che inizia al byte 52.
                                       // Impostare questo valore nel campo testo corrispondente.
                                       int dasd = parm1Converter.toInt (parm1Bytes, 52);
                   dasdField.setText (Integer.toString (dasd / 10000) + "%");

                                       // Richiamare il numero di lavori attivi che inizia al byte 36.
                                       // Impostare questo valore nel campo testo corrispondente.
                                       int jobs = parm1Converter.toInt (parm1Bytes, 36);
                                       jobsField.setText (Integer.toString (jobs));
                }
                                    catch (Exception e) { e.printStackTrace(); }
            }
        });

                    // Effettuare il layout della frame.
                    JPanel outputPanel = new JPanel ();
                    outputPanel.setLayout (new GridLayout (3, 2, 5, 5));
                    outputPanel.add (cpuLabel);
                    outputPanel.add (cpuField);
                    outputPanel.add (dasdLabel);
                    outputPanel.add (dasdField);
                    outputPanel.add (jobsLabel);
                    outputPanel.add (jobsField);

                    Panel buttonPanel = new Panel ();
                    buttonPanel.add (button);

                f.getContentPane ().setLayout (new BorderLayout ());
        f.getContentPane ().add ("Center", outputPanel);
        f.getContentPane ().add ("South", buttonPanel);
                f.pack ();
                f.show ();
    }
                                catch (Exception e)
    {
                         System.out.println ("Error: " + e.getMessage ());
             System.exit(0);
    }
}


}
```

Esempio: utilizzo di VPrinter

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
////////////////////////////////////////////////////////////////////////////
//
// Esempio VPrinter.  Questo programma presenta una stampante ed i relativi file
```

```
// di spool in un pannello explorer.
//
// Sintassi del comando:
//    VPrinterExample system
//
////////////////////////////////////////////////////////////////////////////////

import com.ibm.as400.access.*;
import com.ibm.as400.vaccess.*;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class VPrinterExample
{


        public static void main (String[] args)
    {

                // Se l'utente non fornisce un nome di stampante visualizzare le informazioni sulla stampante
                // per una stampante denominata OS2VPRT;
                String printerName = "OS2VPRT";

                // Se non è stato specificato un sistema, visualizzare il testo di aiuto ed
                // uscire.
                if (args.length == 0)
        {
                        System.out.println("Usage:  VPrinterExample system printer");
                        return;
        }


                // Se l'utente ha specificato un nome, utilizzarlo invece del valore predefinito.
        if (args.length > 1)
                     printerName = args[1];


                                    try
        {
                  // Creare un oggetto AS400.
Il nome di sistema è stato passato
                        // come primo argomento della riga comandi.
                        AS400 system = new AS400 (args[0]);


                        // Creare un oggetto Printer (dal pacchetto access di Toolbox)
                        // che rappresenti la stampante, quindi creare un oggetto
                        // VPrinter per visualizzare graficamente i file di spool nella stampante.
                        Printer   printer = new Printer(system, printerName);
                        VPrinter vprinter = new VPrinter(printer);

                        // Creare una frame per contenere la finestra.
                        JFrame f = new JFrame ("VPrinter Example");

            // Creare un adattatore finestra di dialogo errore.  Questo visualizzerà
                        // qualsiasi errore all'utente.
                        ErrorDialogAdapter errorHandler = new ErrorDialogAdapter (f);


                        // Creare un pannello explorer per presentare la stampante ed i relativi file
            // di spool.  Utilizzare load per caricare le informazioni dal sistema.
                        AS400ExplorerPane explorerPane = new AS400ExplorerPane (vprinter);
                        explorerPane.addErrorListener (errorHandler);
                                explorerPane.load ();
```

```
                        // Quando si chiude la frame, uscire.
                        f.addWindowListener (new WindowAdapter () {
                                public void windowClosing (WindowEvent event)
                {
                          System.exit(0);
                }
            });

                        // Effettuare il layout della frame con il pannello explorer.
                    f.getContentPane ().setLayout (new BorderLayout ());
                        f.getContentPane ().add ("Center", explorerPane);
                    f.pack ();
                    f.show ();
        }
                                    catch (Exception e)
        {
                      System.out.println ("Error: " + e.getMessage ());
                 System.exit(0);
        }
    }
}
```

Esempio: utilizzo di VPrinters

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
////////////////////////////////////////////////////////////////////////
//
// Esempio VPrinters.  Questo programma presenta varie risorse
// stampa di rete con un pannello explorer.
//
// Sintassi del comando:
//    VPrintersExample system
//
// Questo sorgente è un esempio "VPrinters" IBM Toolbox per Java .
//
////////////////////////////////////////////////////////////////////////

import com.ibm.as400.access.*;
import com.ibm.as400.vaccess.*;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class VPrintersExample
{


        public static void main (String[] args)
    {
                // Se non è stato specificato un sistema, visualizzare il testo di aiuto ed
                // uscire.
                if (args.length != 1)
        {
            System.out.println("Usage:  VPrintersExample system");
                        return;
        }

                                    try
        {
                  // Creare un oggetto AS400.
Il nome di sistema è stato passato
                        // come primo argomento della riga comandi.
                        AS400 system = new AS400 (args[0]);

                        // Creare un oggetto VPrinters che rappresenti l'elenco
                        // di stampanti collegate al sistema.
```

```
                    VPrinters printers = new VPrinters (system);

                    // Creare una frame.
            JFrame f = new JFrame ("VPrinters example");

            // Creare un adattatore finestra di dialogo errore.  Questo visualizzerà
                    // qualsiasi errore all'utente.
                    ErrorDialogAdapter errorHandler = new ErrorDialogAdapter (f);

                    // Creare un pannello explorer per presentare le risorse stampa di rete.
                 // Utilizzare load per caricare le informazioni dal sistema.
                    AS400ExplorerPane explorerPane = new AS400ExplorerPane (printers);
                    explorerPane.addErrorListener (errorHandler);
                      explorerPane.load ();

                    // Quando si chiude la frame, uscire.
                    f.addWindowListener (new WindowAdapter () {
                            public void windowClosing (WindowEvent event)
                {
                      System.exit(0);
                }
            });

                    // Effettuare il layout della frame con il pannello explorer.
                   f.getContentPane ().setLayout (new BorderLayout ());
                    f.getContentPane ().add ("Center", explorerPane);
                   f.pack ();
                   f.show ();
        }
                                    catch (Exception e)
        {
                      System.out.println ("Error: " + e.getMessage ());
                 System.exit(0);
        }
    }


}
```

Esempio VPrinterOutput

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
///////////////////////////////////////////////////////////////////////////////
//
// Esempio di VPrinterOutput.  Questo programma presenta un elenco di file
// di spool sul server.  Tutti i file di spool o i file di spool per
// uno specifico utente possono essere visualizzati.
//
// Sintassi del comando:
//    VPrinterOutputExample system <user>
//
//    (User è facoltativo, se non è specificato verranno visualizzati tutti i file di spool
//    sul sistema.  Attenzione - l'elencazione di tutti i file di spool sul sistema
//    può richiedere molto tempo)
//
///////////////////////////////////////////////////////////////////////////////

import com.ibm.as400.access.*;
import com.ibm.as400.vaccess.*;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class VPrinterOutputExample
{
```

```
        public static void main (String[] args)
    {

          // Se non è stato specificato un sistema, visualizzare il testo di aiuto ed uscire.
            if (args.length == 0)
        {
            System.out.println("Usage:  VPrinterOutputExample system <user>");
                        return;
        }

                                    try
        {
                  // Creare un oggetto AS400.
Il nome di sistema è stato passato
                        // come primo argomento della riga comandi.
                        AS400 system = new AS400 (args[0]);
                        system.connectService(AS400.PRINT);

                                  // Creare l'oggetto VPrinterOutput.
                        VPrinterOutput printerOutput = new VPrinterOutput(system);

                        // Se è stato specificato un utente come parametro della riga comandi, indicare
                        // al printerObject di richiamare i file di spool solo per tale utente.
            if (args.length > 1)
                              printerOutput.setUserFilter(args[1]);


                        // Creare una frame per contenere la finestra.
                        JFrame f = new JFrame ("VPrinterOutput Example");

            // Creare un adattatore finestra di dialogo errore.  Questo visualizzerà
                        // qualsiasi errore all'utente.
                        ErrorDialogAdapter errorHandler = new ErrorDialogAdapter (f);


                        // Creare un pannello di dettagli per presentare l'elenco di file di spool.
                 // Utilizzare load per caricare le informazioni dal sistema.
                        AS400DetailsPane detailsPane = new AS400DetailsPane (printerOutput);
                        detailsPane.addErrorListener (errorHandler);
                        detailsPane.load ();


                        // Quando si chiude la frame, uscire.
                        f.addWindowListener (new WindowAdapter () {
                                public void windowClosing (WindowEvent event)
                    {
                              System.exit(0);
                    }
            });

                        // Effettuare il layout della frame con il pannello dettagli.
                    f.getContentPane ().setLayout (new BorderLayout ());
                        f.getContentPane ().add ("Center", detailsPane);
                    f.pack ();
                    f.show ();
        }
                                    catch (Exception e)
        {
                      System.out.println ("Error: " + e.getMessage ());
                 System.exit(0);
        }
    }
}
```

Esempio: utilizzo di SQLQueryBuilderPane

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
///////////////////////////////////////////////////////////////////////
//
// Esempio SQLQueryBuilderPane.  Questo programma presenta un programma di creazione interrogazione
// che consente all'utente di creare un'interrogazione SQL.
//
// Sintassi del comando:
//    SQLQueryBuilderPaneExample system
//
// Questo sorgente è un esempio di "SQLQueryBuilderPane" e
// "SQLResultSetFormPane" di IBM Toolbox per Java.
//
///////////////////////////////////////////////////////////////////////

import com.ibm.as400.access.*;
import com.ibm.as400.vaccess.*;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.sql.*;

public class SQLQueryBuilderPaneExample
{


        // Questo collegamento è condiviso da tutti i componenti.
        private SQLConnection connection;



        // Questo handler dell'errore è condiviso da tutti i componenti.
        private ErrorDialogAdapter errorHandler;



        // Il pannello programma di creazione interrogazione.
        private SQLQueryBuilderPane queryBuilderPane;



    // Questa è la chiamata java principale.  Qui si crea un'istanza della
        // classe e si chiama il metodo Main().  Si effettua questa operazione per evitare
    // problemi con static.  Java ha delle restrizioni rispetto ai metodi
    // static che utilizzano dati non-static, specialmente quando riguardano
        // le classi interne.  Il codice risulta più pulito se si mantiene la quantità di
    // dati e di metodi static ad un livello minimo.
        public static void main (String[] args)
    {
              SQLQueryBuilderPaneExample me = new SQLQueryBuilderPaneExample();
              me.Main(args);
    }


        public void Main (String[] args)
    {
                // Se non è stato specificato un sistema, visualizzare
                // il testo di aiuto ed uscire.
                if (args.length != 1)
        {
            System.out.println("Usage:  SQLQueryBuilderPaneExample system");
                        return;
        }

                                try
```

```
        {
                        // Registrare l'unità di controllo JDBC IBM Toolbox per Java.
                        DriverManager.registerDriver (new AS400JDBCDriver ());

                                    // Creare un oggetto SQLConnection.
Il nome di sistema è stato passato
                        // come primo argomento della riga comandi.
            connection = new SQLConnection ("jdbc:as400://" + args[0]);

                        // Creare una frame.
            JFrame f = new JFrame ("SQLQueryBuilderPane example");

            // Creare un adattatore finestra di dialogo errore.  Questo visualizzerà
                        // qualsiasi errore all'utente.
                        errorHandler = new ErrorDialogAdapter (f);

            // Creare un pannello progr. di creazione interrogazione SQL per presentare il programma
            // di creazione dell'interrogazione.
Caricare i dati necessari per la
                        // interrogazione dal sistema.
                        queryBuilderPane = new SQLQueryBuilderPane (connection);
                        queryBuilderPane.addErrorListener (errorHandler);
                        queryBuilderPane.load ();

                        // Creare un pulsante che visualizzerà i risultati
                        // dell'interrogazione creata in un pannello modulo in un'altra frame.
            JButton resultSetButton = new JButton ("Show result set");
                        resultSetButton.addActionListener (new ActionListener ()
            {
                                public void actionPerformed (ActionEvent event)
                {
                                        showFormPane (queryBuilderPane.getQuery ());
                }
            });

                        // Quando si chiude la frame, uscire.
                        f.addWindowListener (new WindowAdapter ()
            {
                                public void windowClosing (WindowEvent event)
                {
                                  System.exit(0);
                }
            });

            // Effettuare il layout della frame con il pannello progr. di creazione interrogazione.
            f.getContentPane ().setLayout (new BorderLayout ());
            f.getContentPane ().add ("Center", queryBuilderPane);
            f.getContentPane ().add ("South", resultSetButton);
                    f.pack ();
                    f.show ();
        }
                                    catch (Exception e)
        {
                      System.out.println ("Error: " + e.getMessage ());
                 System.exit(0);
        }
    }



        private void showFormPane (String query)
    {
                // Creare una nuova frame per i risultati dell'interrogazione.
                JFrame f = new JFrame (query);

                // Creare un pannello modulo della serie di risultati SQL per presentare i risultati
                // dell'interrogazione.  Caricare i risultati dal sistema.
```

```
        SQLResultSetFormPane formPane = new SQLResultSetFormPane (connection, query);
                formPane.addErrorListener (errorHandler);
                formPane.load ();

                // Effettuare il layout della frame con il pannello modulo.
                f.getContentPane ().setLayout (new BorderLayout ());
        f.getContentPane ().add ("Center", formPane);
                f.pack ();
                f.show ();
    }


}
```

Esempio: utilizzo di SQLResultSetTablePane

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
////////////////////////////////////////////////////////////////////////
//
// Esempio SQLResultSetTablePane.  Questo programma presenta il contenuto di
// una tabella in un pannello di tabella.  Vi è SQLStatementDocument che consente
// all'utente di immettere qualsiasi istruzione SQL.  Inoltre, vi è un pulsante
// che consente all'utente di cancellare tutte le righe della tabella.
//
// Sintassi del comando:
//    SQLResultSetTablePaneExample  system  table
//
// Questo sorgente è un esempio di "SQLQueryBuilderPane",
// "SQLResultSetFormPane" e "SQLStatementButton" di IBM Toolbox per Java .
//
////////////////////////////////////////////////////////////////////////

import com.ibm.as400.access.*;
import com.ibm.as400.vaccess.*;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.sql.*;

public class SQLResultSetTablePaneExample
{



        private static SQLStatementDocument     document;
        private static SQLResultSetTablePane    tablePane;



        public static void main (String[] args)
    {
                // Se non è stato specificato un sistema, visualizzare
                // il testo di aiuto ed uscire.
                if (args.length != 2)
        {
            System.out.println("Usage:  SQLResultSetTablePaneExample  system  table");
                        return;
        }

                                    try
        {
                        // Registrare l'unità di controllo JDBC IBM Toolbox per Java.
                        DriverManager.registerDriver (new AS400JDBCDriver ());

                                   // Creare un oggetto SQLConnection.
Il nome di sistema è stato passato
```

```
                        // come primo argomento della riga comandi.
Questo collegamento è
                        // condiviso da tutti i componenti.
            SQLConnection connection = new SQLConnection ("jdbc:as400://" + args[0]);

                        // Creare una frame.
            JFrame f = new JFrame ("SQLResultSetTablePane example");

            // Creare un adattatore finestra di dialogo errore.  Questo visualizzerà
                        // qualsiasi errore all'utente.
Questo handler dell'errore è condiviso
                        // da tutti i componenti.
                        ErrorDialogAdapter errorHandler = new ErrorDialogAdapter (f);

                        // Creare un documento istruzione SQL che consenta
                        // all'utente di immettere un'interrogazione.
            document = new SQLStatementDocument (connection, "");
                        document.addErrorListener (errorHandler);

                        // Creare un campo di testo per la presentazione del documento.
                        JTextField textField = new JTextField (document,
                "Enter a SQL statement here.", 50);

                        // Creare un pulsante che cancelli tutte le righe della tabella.
            SQLStatementButton deleteAllButton = new SQLStatementButton ("Delete all rows");
                        deleteAllButton.setConnection (connection);
            deleteAllButton.setSQLStatement ("DELETE FROM " + args[1]);
                        deleteAllButton.addErrorListener (errorHandler);

                        // Creare un pannello tabella serie di risultati SQL per presentare i risultati
            // di un'interrogazione.  Caricare immediatamente il contenuto.
            tablePane = new SQLResultSetTablePane (connection, "SELECT * FROM " + args[1]);
                        tablePane.addErrorListener (errorHandler);
                                tablePane.load ();

                        // Quando si preme Invio nel campo testo,
                        // eseguire l'istruzione SQL ed aggiornare il pannello tabella.
                        textField.addKeyListener (new KeyAdapter ()
            {
                                public void keyPressed (KeyEvent event)
                    {
                                        if (event.getKeyCode () == KeyEvent.VK_ENTER)
                            {
                                                // Se l'istruzione SQL è SELECT,
                                                // lasciare che il pannello tabella la esegua, altrimenti,
                                                // lasciare che la esegua il documento.
                                                String sql = document.getSQLStatement ();
                                if (sql.toUpperCase ().startsWith ("SELECT"))
                                {
                                                        try
                                    {
                                                                tablePane.setQuery (sql);
                                    }
                                                        catch (Exception e)
                                    {
                                                                // Ignorare.
                                    }
                                                tablePane.load ();
                                }
                                                else
                                                        document.execute ();
                            }
                    }
            });

                        // Una volta cancellate tutte le righe utilizzando il pulsante,
                        // aggiornare il pannello tabella.
```

```
                deleteAllButton.addActionCompletedListener (new ActionCompletedListener ()
        {
                        public void actionCompleted (ActionCompletedEvent event)
            {
                                tablePane.load ();
            }
        });

                // Quando si chiude la frame, uscire.
                f.addWindowListener (new WindowAdapter ()
        {
                        public void windowClosing (WindowEvent event)
            {
                  System.exit(0);
            }
        });

        // Effettuare il layout della frame con il pannello progr. di creazione interrogazione.
        f.getContentPane ().setLayout (new BorderLayout ());
        f.getContentPane ().add ("North", textField);
        f.getContentPane ().add ("Center", tablePane);
        f.getContentPane ().add ("South", deleteAllButton);
                f.pack ();
                f.show ();
    }
                                catch (Exception e)
    {
                  System.out.println ("Error: " + e.getMessage ());
             System.exit(0);
    }
}


}
```

Esempi: XPCML

Questa sezione elenca gli esempi di codice forniti in tutta la documentazione del componente XPCML di IBM Toolbox per Java.

- “Esempio: condensamento di un documento XPCML esistente” a pagina 752
- “Esempio: condensamento di un documento XPCML esistente, incluso il codice Java” a pagina 753
- “Esempio: utilizzo di XPCML condensato per creare un oggetto ProgramCallDocument” a pagina 755
- “Esempio: come ottenere i risultati della chiamata al programma come XPCML condensato” a pagina 756
- “Esempio: richiamo dei risultati di una chiamata al programma come XPCML” a pagina 747
- “Esempio: inoltro dei valori di parametro come XPCML” a pagina 749
- “Esempi: inoltro delle schiere di valori di parametro come XPCML” a pagina 751
- “Esempio: conversione di un documento PCML in un documento XPCML” a pagina 442

Il seguente esonero di responsabilità si applica a tutti gli esempi IBM Toolbox per Java:

Esonero di responsabilità per gli esempi di codice

L’IBM fornisce una licenza non esclusiva per utilizzare tutti gli esempi del codice di programmazione da cui creare funzioni simili personalizzate, in base a richieste specifiche.

Questo codice di esempio è fornito dall'IBM con la sola funzione illustrativa. Questi esempi non sono stati interamente testati in tutte le condizioni. IBM, perciò, non fornisce nessun tipo di garanzia o affidabilità implicita, rispetto alla funzionalità o alle funzioni di questi programmi.

Tutti i programmi qui contenuti vengono forniti all'utente ″COSI' COME SONO″ senza garanzie di alcun tipo. Le garanzie implicite di non contraffazione, commerciabilità e adeguatezza a scopi specifici sono espressamente vietate.

Esempio: richiamo dei risultati di una chiamata al programma come XPCML

Il seguente esempio mostra come è possibile creare un ProgramCallDocument XPCML, chiamare un programma iSeries e richiamare i risultati della chiamata al programma come XPCML. L'esempio presuppone i seguenti componenti:

- Documento XPCML qgyolaus.xpcml, che definisce le specifiche di programma e parametro con valori di immissione
- Codice Java che crea un oggetto ProgramCallDocument, utilizza il file XPCML e quindi chiama il programma QGYOLAUS
- Risultati della chiamata al programma, che il codice Java genera come XPCML e memorizza nel file XPCMLOut.xpcml

Si noti come i dati della schiera sono specificati nell'XPCML originale e generato. L'elemento qgyolaus.receiver, un parametro di emissione, è un arrayOfStructParm XPCML con un attributo che imposta il conteggio su listInfo.rcdsReturned. Il seguente codice di esempio include solo parte dell'emissione QGYOLAUS. Se l'esempio includesse tutta l'emissione, il codice potrebbe elencare 89 utenti sotto la tag XPCML <arrayOfStructParm>.

Per schiere di struct, XPCML utilizza la tag XPCML <struct_i> per delimitare ogni elemento structParm. Ogni tag <struct_i> indica che i dati racchiusi in essa sono un elemento di tipo autu0150 struct. L'attributo indice della tag <struct_i> specifica l'elemento della per la struct.

Per schiere di tipi semplici, come ad esempio arrayOfStringParm, arrayOfIntParm e così via, la tag XPCML <i> elenca gli elementi della schiera.

Documento XPCML qgyolaus.xpcml

```
<xpcml xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:noNamespaceSchemaLocation="xpcml.xsd" version="4.0">

<!-- XPCML source for calling "Open List of Authorized Users" -->
<!-- (QGYOLAUS) API -->

  <!-- Format AUTU0150 - Other formats are available -->
  <struct name="autu0150">
    <stringParm name="name" length="10"/>
    <stringParm name="userOrGroup" length="1"/>
    <stringParm name="groupMembers" length="1"/>
    <stringParm name="description" length="50"/>
  </struct>

  <!-- List information structure (common for "Open List" type APIs) -->
  <struct name="listInfo">
    <intParm name="totalRcds"/>
    <intParm name="rcdsReturned">0</rcdsReturned>
    <hexBinaryParm name="rqsHandle" totalBytes="4"/>
    <intParm name="rcdLength"/>
    <stringParm name="infoComplete" length="1"/>
    <stringParm name="dateCreated" length="7"/>
    <stringParm name="timeCreated" length="6"/>
    <stringParm name="listStatus" length="1"/>
    <hexBinaryParm totalBytes="1"/>
```

```
  <unsignedIntParm name="lengthOfInfo"/>
  <intParm name="firstRecord"/>
  <hexBinaryParm totalBytes="40"/>
</struct>

<!-- Program QGYOLAUS and its parameter list for retrieving -->
<!-- AUTU0150 format -->

<program name="QGYOLAUS" path="/QSYS.lib/QGY.lib/QGYOLAUS.pgm"
   parseOrder="listInfo receiver">
 <parameterList>
   // Valori di emissione --- schiera di autu0150 struct
   <arrayOfStructParm name="receiver" count="listInfo.rcdsReturned"
      passDirection="out" outputSize="receiverLength" struct="autu0150"/>
   // Valori di immissione
   <intParm name="receiverLength" passDirection="in">16384</intParm>
   <structParm name="listInfo" passDirection="out" struct="listInfo"/>
   // Valori di immissione
   <intParm name="rcdsToReturn" passDirection="in">264</intParm>
   <stringParm name="format" passDirection="in" length="10">
      AUTU0150</stringParm>
   <stringParm name="selection" passDirection="in" length="10">
      *USER</stringParm>
   <stringParm name="member" passDirection="in" length="10">
      *NONE</stringParm>
   <intParm name="errorCode" passDirection="in">0</intParm>
 </parameterList>
</program>
```

Codice Java che crea l'oggetto ProgramCallDocument e utilizza XPCML per chiamare il programma QGYOLAUS

```
system = new AS400();
// Creare ProgramCallDocument in cui analizzare il file.
ProgramCallDocument xpcmlDoc =
   new ProgramCallDocument(system, "QGYOLAUS.xpcml");

// Chiamare QGYOLAUS
boolean rc = xpcmlDoc.callProgram("QGYOLAUS");

// Ottenere i risultati della chiamata al programma come XPCML e memorizzarli
// nel file XPCMLOut.xpcml
if (rc)     // Il programma ha avuto esito positivo
   xpcmlDoc.generateXPCML("QGYOLAUS","XPCMLOut.xpcml");
```

Risultati della chiamata al programma, generati come XPCML e memorizzati nel file XPCMLOut.xpcml

```
<?xml version="1.0"?>
<xpcml version="4.0"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:noNamespaceSchemaLocation='xpcml.xsd' >

  <program name="QGYOLAUS" path="/QSYS.lib/QGY.lib/QGYOLAUS.pgm"
     parseOrder="listInfo receiver">
  <parameterList>
      <arrayOfStructParm name="receiver" passDirection="out"
          count="listInfo.rcdsReturned" outputSize="receiverLength"
          struct="autu0150">
        <struct_i index="0">
          <stringParm name="name" length="10">JANEDOW</stringParm>
          <stringParm name="userOrGroup" length="1">0</stringParm>
          <stringParm name="groupMembers" length="1">0</stringParm>
          <stringParm name="description" length="50">
             Jane Doe</stringParm>
        </struct_i>
        <struct_i index="1">
```

```
          <stringParm name="name" length="10">BOBS</stringParm>
          <stringParm name="userOrGroup" length="1">0</stringParm>
          <stringParm name="groupMembers" length="1">0</stringParm>
          <stringParm name="description" length="50">
             Bob Smith</stringParm>
        </struct_i>

        <!-- More records here depending on how many users output. -->
        <!-- In this case 89 user records are listed here.    -->

      </arrayOfStructParm>     <!-- End of user array -->
      <intParm name="receiverLength" passDirection="in">
         16384</intParm>
      <structParm name="listInfo" passDirection="out"
         struct="listInfo">
          <intParm name="totalRcds">89</intParm>
          <intParm name="rcdsReturned">89</intParm>
          <hexBinaryParm name="rqsHandle" totalBytes="4">
             00000001==</hexBinaryParm>
          <intParm name="rcdLength">62</intParm>
          <stringParm name="infoComplete" length="1">C</stringParm>
          <stringParm name="dateCreated" length="7">
             1030321</stringParm>
          <stringParm name="timeCreated" length="6">
             120927</stringParm>
          <stringParm name="listStatus" length="1">2</stringParm>
          <hexBinaryParm totalBytes="1"></hexBinaryParm>
          <unsignedIntParm name="lengthOfInfo">
             5518</unsignedIntParm>
          <intParm name="firstRecord">1</intParm>
      </structParm>
      <intParm name="rcdsToReturn" passDirection="in">264</intParm>
      <stringParm name="format" passDirection="in" length="10">
         AUTU0150</stringParm>
      <stringParm name="selection" passDirection="in" length="10">
         *USER</stringParm>
      <stringParm name="member" passDirection="in" length="10">
         *NONE</stringParm>
      <intParm name="errorCode" passDirection="in">0</intParm>
    </parameterList>
    </program>
  </xpcml>
```

Esempio: inoltro dei valori di parametro come XPCML

Valori dei parametri del programma possono essere impostati nel file sorgente XPCML. Quando XPCML viene letto e analizzato, il metodo setValue di ProgramCallDocument viene chiamato automaticamente per ogni parametro il cui valore è stato trasmesso come XPCML. Questo esonera l'utente dal dover scrivere il codice Java per impostare i valori di strutture e schiere complicate.

Nei seguenti esempi, l'XPCML chiama due programmi differenti, prog1 e prog2. Entrambi i programmi utilizzano il parametro di immissione s1Ref. Il primo esempio imposta valori differenti per s1Ref per ogni chiamata al programma. Il secondo esempio specifica lo stesso valore per s1Ref per ogni chiamata al programma, che illustra un modo utile per impostare valori di dati costanti per parametri di immissione.

Esempio: inoltro di differenti per i parametri di immissione

Nel seguente esempio, dopo che il programma di analisi XML legge ed analizza il documento, il valore dell'elemento prog1.s1Ref.s2Ref.s2p1[0] è prog1Val_1 ed il valore dell'elemento prog1.s1Ref.s2Ref.s2p1[1] è prog1Val_2.

```
<xpcml xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="xpcml.xsd" version="4.0">
```

```
<struct name="s1">
   <stringParm name="s1p1"/>
   <structParm name="s2Ref" struct="s2"/>
</struct>

<struct name="s2">
  <stringParm name="s2p1" length="10"/>
  <arrayOfStringParm name="parm1" count="2"/>
</struct>

<program name="prog1" path="/QSYS.LIB/W95LIB.LIB/PROG1.PGM">
 <parameterList>
   <structParm name="s1Ref" struct="s1" passDirection="in" >
     <stringParm name="s1p1">prog1Val</stringParm>
     <structParm name="s2Ref" struct="s2">
         <stringParm name="s2p1" length="10">prog1Val</stringParm>
         <arrayOfStringParm name="parm1" count="2">
            <i>prog1Val_1</i>
            <i>prog1Val_2</i>
         </arrayOfStringParm>
     </structParm>
   </structParm>
 </parameterList>
</program>

<program name="prog2" path="/QSYS.LIB/W95LIB.LIB/PROG1.PGM">
 <parameterList>
   <structParm name="s1Ref" struct="s1" passDirection="in" >
     <stringParm name="s1p1">prog2Val</stringParm>
     <structParm name="s2Ref" struct="s2">
         <stringParm name="s2p1" length="10">prog2Val</stringParm>
         <arrayOfStringParm name="parm1" count="2">
            <i>prog2Val_1</i>
            <i>prog2Val_2</i>
         </arrayOfStringParm>
     </structParm>
   </structParm>
 </parameterList>
</program>
</xpcml>
```

Esempio: inoltro di valori costanti per i parametri di immissione

Nel seguente esempio, dopo che il programma di analisi XML legge ed analizza il documento, il valore dell'elemento prog1.s1Ref.s2Ref.s2p1[0] è constantVal_1 ed il valore dell'elemento prog1.s1Ref.s2Ref.s2p1[1] è constantVal_2.

```
<xpcml xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:noNamespaceSchemaLocation="xpcml.xsd" version="4.0">

<struct name="s1" >
   <stringParm name="s1p1">constantVal</stringParm>
   <structParm name="s2Ref" struct="s2"/>
</struct>

<struct name="s2">
  <stringParm name="s2p1" length="10">constantVal</stringParm>
  <arrayOfStringParm name="parm1" count="2">
     <i>constantVal_1</i>
     <i>constantVal_2</i>
  </arrayOfStringParm>
</struct>

<program name="prog1" path="/QSYS.LIB/W95LIB.LIB/PROG1.PGM">
 <parameterList>
   <structParm name="s1Ref" struct="s1" passDirection="in" />
 </parameterList>
```

```
 </program>

 <program name="prog2" path="/QSYS.LIB/W95LIB.LIB/PROG1.PGM">
  <parameterList>
    <structParm name="s1Ref" struct="s1" passDirection="in" />
  </parameterList>
 </program>
</xpcml>
```

Esempi: inoltro delle schiere di valori di parametro come XPCML

Quando si utilizza XPCML per trasmettere dati di schiera, è necessario utilizzare l'attributo di conteggio:

- Specificare l'attributo conteggio nell'elemento schiera
- Impostare l'attributo conteggio sul numero di elementi che la schiera contiene al momento dell'analisi del documento

Il seguente esempio illustra come trasmettere schiere di valori parametro utilizzando i dati di schiera structParm ed una schiera di structs.

```
<?xml version="1.0" encoding="UTF-8"?>
<xpcml xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:noNamespaceSchemaLocation="xpcml.xsd" version="4.0">

 <struct name="s1" >
    <stringParm name="s1p1"/>
    <struct name="s1Array">
      <stringParm name="s1Ap1"/>
    </struct>
 </struct>

 <struct name="s2">
    <stringParm name="s2p1"/>
 </struct>

 <program name="prog1" path="/QSYS.LIB/W95LIB.LIB/PROG1.PGM">
  <parameterList>
    <structParm name="s1Ref" struct="s1" passDirection="in" >
        <stringParm name="s1p1">Value 1</stringParm>
        <arrayOfStruct name="s1Array" count="2">
           <struct_i>
               <stringParm name="s1Ap1">Value 1</stringParm>
           </struct_i>
           <struct_i>
               <stringParm name="s1Ap1">Value 2</stringParm>
           </struct_i>
         </arrayOfStruct>
    </structParm>
    <arrayOfStructParm name="s2Ref" struct="s2" count="2" passDirection="in" >
      <struct_i>
          <stringParm name="s2p1">Value 1</stringParm>
      </struct_i>
      <struct_i>
          <stringParm name="s2p1">Value 2</stringParm>
      </struct_i>
    </arrayOfStructParm>
  </parameterList>
 </program>
</xpcml>
```

Ad esempio, il seguente XPCML specifica una schiera di 3 intParms ed imposta il primo elemento su 12, il secondo su 100 ed il terzo su 4:

```
<?xml version="1.0"?>
<xpcml version="4.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:noNamespaceSchemaLocation='xpcml.xsd' >
```

```
  <program name="prog1" path="/QSYS.lib/MYLIG.lib/PROG1.pgm">
  <parameterList>
    <arrayOfIntParm name="intArray" count="3">
      <i>12</i>
      <i>100</i>
      <i>4</i>
    </arrayOfIntParm>
  </parameterList>
  </program>
</xpcml>
```

Utilizzo dell'attributo indice delle tag <i> e <struct_i> per impostare i valori di schiera

E' possibile utilizzare l'attributo indice delle tag <i> e <struct_i> per aiuto nell'impostazione dei valori di schiera. Nel seguente esempio, l'XPCML imposta il primo elemento della schiera su 4, il secondo su 100 e il terzo su 12.

```
<?xml version="1.0"?>
<xpcml version="4.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:noNamespaceSchemaLocation='xpcml.xsd' >

  <program name="prog1" path="/QSYS.lib/MYLIG.lib/PROG1.pgm">
  <parameterList>
    <arrayOfIntParm name="intArray" count="3">
      <i index="2">12</i>
      <i index="1">100</i>
      <i index="0">4</i>
    </arrayOfIntParm>
  </parameterList>
  </program>
</xpcml>
```

Esempio: condensamento di un documento XPCML esistente

Il seguente esempio illustra come condensare un documento XPCML esistente. L'esempio include il sorgente XPCML originale, l' XPCML condensato risultante e lo schema esteso.

Sorgente XPCML originale

```
<?xml version="1.0" encoding="UTF-8"?>
 <xpcml xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xsi:noNamespaceSchemaLocation="xpcml.xsd" version="4.0">
  <program name="prog1" path="/QSYS.LIB/W95LIB.LIB/PROG1.PGM">
    <parameterList>
       <stringParm name="parm1" passDirection="in" passMode="value"
             minvrm="V5R2M0" ccsid="37" length="10">Value 1</stringParm>
    </parameterList>
  </program>
 </xpcml>
```

Sorgente XPCML condensato

```
<?xml version="1.0" encoding="UTF-8"?>
<xpcml xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:noNamespaceSchemaLocation="myXSD.xsd" version="4.0">
 <program name="prog1" path="/QSYS.LIB/W95LIB.LIB/PROG1.PGM">
    <parameterList>
       <parm1_>Value 1</parm1_>
    </parameterList>
 </program>
</xpcml>
```

Schema creato

```
<!-- parm1's XSD definition -->
<xs:schema xmlns:xs='http://www.w3.org/2001/XMLSchema'>
 <!-- Link back to XPCML.xsd -->
 <xs:include schemaLocation='xpcml.xsd'/>
  <xs:element name="parm1_" substitutionGroup="stringParmGroup" >
   <xs:complexType>
     <xs:simpleContent>
     <xs:restriction base="stringParmType">
        <!-- Attributes defined for parm1 -->
        <xs:attribute name="name" type="string50" fixed="parm1" />
        <xs:attribute name="length" type="xs:string" fixed="10" />
        <xs:attribute name="passMode" type="xs:string" fixed="value" />
        <xs:attribute name="ccsid" type="xs:string" fixed="37" />
        <xs:attribute name="minvrm" type="xs:string" fixed="V5R2M0" />
     </xs:restriction>
     </xs:simpleContent>
   </xs:complexType>
  </xs:element>
</schema>
```

Esempio: condensamento di un documento XPCML esistente, incluso il codice Java

Il seguente esempio illustra come condensare un documento XPCML esistente. L'esempio include il sorgente XPCML originale, l'XPCML risultante condensato, il codice Java che chiama condenseXPCML() e qualche definizione tipo appena creata nello schema esteso:

Sorgente XPCML originale

```
<!-- Fully specified XPCML source -->
<xpcml xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:noNamespaceSchemaLocation="xpcml.xsd" version="4.0">

<struct name="qualifiedJobName">
   <stringParm name="jobName" length="10">*</stringParm>
   <stringParm name="userName" length="10"/>
   <stringParm name="jobNumber" length="6"/>
</struct>

<struct name="jobi0100">
   <intParm name="numberOfBytesReturned"/>
   <intParm name="numberOfBytesAvailable"/>
   <structParm name="qualifiedJobName" struct="qualifiedJobName"/>
   <hexBinaryParm name="internalJobIdentifier" totalBytes="16"/>
   <stringParm name="jobStatus" length="10"/>
   <stringParm name="jobType" length="1"/>
   <stringParm name="jobSubtype" length="1"/>
   <stringParm length="2"/>
   <intParm name="runPriority"/>
   <intParm name="timeSlice"/>
   <intParm name="defaultWait"/>
   <stringParm name="purge" length="10"/>
</struct>

<program name="qusrjobi_jobi0100" path="/QSYS.LIB/QUSRJOBI.PGM">
   <parameterList>
      <structParm name="receiverVariable" passDirection="out"
         outputSize="lengthOfReceiverVariable" struct="jobi0100"/>
      <intParm name="lengthOfReceiverVariable" passDirection="in">86</intParm>
      <stringParm name="formatName" passDirection="in" length="8">JOBI0100</stringParm>
      <structParm name="qualifiedJobName" passDirection="in" struct="qualifiedJobName"/>
      <hexBinaryParm name="internalJobIdentifier"
         passDirection="in" totalBytes="16"> </hexBinaryParm>
```

```
      <intParm name="errorCode" passDirection="in">0</intParm>
   </parameterList>
</program>
</xpcml>
```

Codice Java per condensare il sorgente XPCML originale

```
        try {
   FileInputStream fullStream = new FileInputStream("myXPCML.xpcml");
   FileOutputStream condensedStream = new FileOutputStream("myCondensedXPCML.xpcml");
   FileOutputStream xsdStream = new FileOutputStream("myXSD.xsd");
   ProgramCallDocument.condenseXPCML(fullStream, xsdStream, condensedStream, "myXSD.xsd");

}
        catch (Exception e) {
  System.out.println("error:  - "+e.getMessage());
          e.printStackTrace();
}
```

Sorgente XPCML condensato: myCondensedXPCML.xpcml

```
<xpcml xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:noNamespaceSchemaLocation="myXSD.xsd" version="4.0">

<struct name="qualifiedJobName">
   <jobName_>*</jobName_>
   <userName_/>
   <jobNumber_/>
</struct>

<struct name="jobi0100">
   <numberOfBytesReturned_/>
   <numberOfBytesAvailable_/>
   <structParm name="qualifiedJobName" struct="qualifiedJobName"/>
   <internalJobIdentifier_/>
   <jobStatus_/>
   <jobType_/>
   <jobSubtype_/>
   <stringParm length="2"/>
   <runPriority_/>
   <timeSlice_/>
   <defaultWait_/>
   <purge_/>
</struct>

<program name="qusrjobi_jobi0100" path="/QSYS.LIB/QUSRJOBI.PGM">
   <parameterList>
      <structParm name="receiverVariable" passDirection="out"
         outputSize="lengthOfReceiverVariable" struct="jobi0100"/>
      <lengthOfReceiverVariable_>86</lengthOfReceiverVariable_>
      <formatName_>JOBI0100</formatName_>
      <structParm name="qualifiedJobName" passDirection="in" struct="qualifiedJobName"/>
      <internalJobIdentifier_> </internalJobIdentifier_>
      <errorCode_>0</errorCode_>
   </parameterList>
</program>
</xpcml>
```

Alcune definizioni tipo dallo schema creato: myXSD.xsd

```
<xs:schema xmlns:xs='http://www.w3.org/2001/XMLSchema'>
<xs:include schemaLocation='xpcml.xsd'/>

<xs:element name="jobName_" substitutionGroup="stringParmGroup" >
   <xs:complexType>
      <xs:simpleContent>
      <xs:restriction base="stringParmType">
         <xs:attribute name="name" type="string50" fixed="jobName" />
```

```
         <xs:attribute name="length" type="xs:string" fixed="10" />
      </xs:restriction>
      </xs:simpleContent>
   </xs:complexType>
</xs:element>

<xs:element name="userName_" substitutionGroup="stringParmGroup" >
   <xs:complexType>
      <xs:simpleContent>
      <xs:restriction base="stringParmType">
         <xs:attribute name="name" type="string50" fixed="userName" />
         <xs:attribute name="length" type="xs:string" fixed="10" />
      </xs:restriction>
      </xs:simpleContent>
   </xs:complexType>
</xs:element>

<xs:element name="jobNumber_" substitutionGroup="stringParmGroup" >
   <xs:complexType>
      <xs:simpleContent>
      <xs:restriction base="stringParmType">
         <xs:attribute name="name" type="string50" fixed="jobNumber" />
         <xs:attribute name="length" type="xs:string" fixed="6" />
      </xs:restriction>
      </xs:simpleContent>
   </xs:complexType>
</xs:element>

<xs:element name="lengthOfReceiverVariable_" substitutionGroup="intParmGroup" >
   <xs:complexType>
      <xs:simpleContent>
      <xs:restriction base="intParmType">
         <xs:attribute name="name" type="string50" fixed="lengthOfReceiverVariable" />
         <xs:attribute name="passDirection" type="passDirectionType" fixed="in" />
      </xs:restriction>
      </xs:simpleContent>
   </xs:complexType>
</xs:element>

<xs:element name="formatName_" substitutionGroup="stringParmGroup" >
   <xs:complexType>
      <xs:simpleContent>
      <xs:restriction base="stringParmType">
         <xs:attribute name="name" type="string50" fixed="formatName" />
         <xs:attribute name="length" type="xs:string" fixed="8" />
         <xs:attribute name="passDirection" type="passDirectionType" fixed="in" />
      </xs:restriction>
      </xs:simpleContent>
   </xs:complexType>
</xs:element>

<!-- More type definitions for each newly defined type follow here -->
</xs:schema>
```

Esempio: utilizzo di XPCML condensato per creare un oggetto ProgramCallDocument

Alcuni programmi di creazione ProgramCallDocument accettano un file sorgente condensedXPCML ed il corrispondente schema (.xsd file). Questo consente all'utente di utilizzare XPCML condensato per creare un oggetto ProgramCallDocument.

I programmi di creazione precedentemente menzionati richiedono all'utente di fornire i seguenti parametri:

- Una Stringa che specifichi un file XPCML condensato

- Un InputStream che contenga le definizioni tipo create eseguendo condenseXPCML()

Utilizzando questi programmi di creazione si carica e si analizza un file XPCML condensato. Inoltre, il processo registra qualsiasi errore di analisi. Una volta completata l'analisi, il programma di creazione crea un oggetto ProgramCallDocument.

Il seguente esempio di codice Java utilizza XPCML condensato per creare un oggetto ProgramCallDocument. Il codice di esempio presuppone quanto segue

- Il nome del file condensato XPCML è myCondensedXPCML.xpcml
- Il nome dello schema esteso è myXSD.xsd

Il codice quindi utilizza l'oggetto ProgramCallDocument per eseguire il programma qusrjobi_jobi0100.

```
          AS400 system = new AS400();
// Creare ProgramCallDocument ed analizzare il file.
ProgramCallDocument xpcmlDoc =
  new ProgramCallDocument(system,
                          "myCondensedXPCML.xpcml",
                          new FileInputStream("myXSD.xsd"));
boolean rc = xpcmlDoc.callProgram("qusrjobi_jobi0100");
```

Nota: il codice XPCML che si utilizza per chiamare il programma (dopo aver creato l'oggetto ProgramCallDocument) è uguale al codice che si utilizzerebbe con PCML.

Esempio: come ottenere i risultati della chiamata al programma come XPCML condensato

Si utilizza lo stesso processo per ottenere i risultati di una chiamata di programma che si utilizza con XPCML condensato o non condensato. Tutto ciò che si deve fare è chiamare ProgramCallDocument.generateXPCML().

Utilizzare setXsdName() per specificare il nome dello schema esteso, che generateXPCML() utilizza per creare l'attributo noNamespaceSchemaLocation della tag <xpcml> nell'XPCML condensato.

L'utilizzo di setXsdName() è importante quando si desidera utilizzare i risultati della chiamata al programma (in XPCML condensato) come sorgente per un altro oggetto ProgramCallDocument. E' necessario specificare il nome dello schema esteso in modo che il programma di analisi sappia quale file di schema utilizzare durante l'analisi.

Ad esempio, il seguente codice ottiene i risultati da una chiamata al programma e genera XPCML condensato.

```
          AS400 system = new AS400();

// Creare ProgramCallDocument ed analizzare il file.
ProgramCallDocument xpcmlDoc =
   new ProgramCallDocument(system, "myCondensedXPCML.xpcml", new FileInputStream("myXSD.xsd"));

boolean rc = xpcmlDoc.callProgram("qusrjobi_jobi0100");

if (rc)     // Il programma ha avuto esito positivo
{
  xpcmlDoc.setXsdName("myXSD.xsd");
  xpcmlDoc.generateXPCML("qusrjobi_jobi0100","XPCMLOut.xpcml");
}
```

Il seguente codice mostra un esempio di come ottenere i risultati di una chiamata al programma come XPCML condensato:

```
<xpcml xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="myXSD.xsd" version="4.0">
```

```
<program name="qusrjobi_jobi0100" path="/QSYS.LIB/QUSRJOBI.PGM">
   <parameterList>
   <structParm name="receiverVariable" passDirection="out"
       outputSize="lengthOfReceiverVariable" struct="jobi0100"/>
            <numberOfBytesReturned_>100</numberOfBytesReturned_>
            <numberOfBytesAvailable_>100</numberOfBytesAvailable_>
            <structParm name="qualifiedJobName"
                struct="qualifiedJobName">
                <jobName_>*</jobName_>
                <userName_/>
                <jobNumber_/>
            </structParm>
            <internalJobIdentifier_/>
            <jobStatus_>ACTIVE</jobStatus_>
            <jobType_>PJ</jobType>
            <jobSubtype_/>
            <stringParm length="2"/>
            <runPriority_>5</runPriority>
            <timeSlice_/>
            <defaultWait_>10</defaultWait_>
            <purge_/>
        </structParm>
        <lengthOfReceiverVariable_>86</lengthOfReceiverVariable_>
        <formatName_>JOBI0100</formatName_>
        <structParm name="qualifiedJobName" passDirection="in" struct="qualifiedJobName"/>
        <internalJobIdentifier_> </internalJobIdentifier_>
        <errorCode_>0</errorCode_>
</parameterList>
</program>
</xpcml>
```

Informazioni correlate per IBM Toolbox per Java

L'elenco seguente include siti Web e argomenti dell'Information Center relativi alle informazioni su IBM Toolbox per Java.

Risorse di IBM Toolbox per Java

Utilizzare i seguenti siti per acquisire ulteriori informazioni su IBM Toolbox per Java:

- IBM Toolbox per Java e JTOpen : offre informazioni sui service pack, suggerimenti sulle prestazioni, esempi ed altro ancora.E' possibile anche scaricare un pacchetto compresso di tali informazioni, inclusi i javadoc.
- FAQ (Frequently Asked Questions) di IBM Toolbox per Java. : fornisce risposte a domande sulle prestazioni, sulla risoluzione di problemi, JDBC ed altro ancora.
- IBM Toolbox per Java e forum JTOpen : offre un metodo efficace per comunicare con la comunità dei programmatori Java che utilizzano IBM Toolbox per Java e con gli stessi sviluppatori di IBM Toolbox per Java.

Risorse di IBM Toolbox per Java 2 Micro Edition

Utilizzare i seguenti siti per acquisire ulteriori informazioni su ToolboxME per iSeries e sull'implementazione Java delle tecnologie wireless:

- IBM Toolbox per Java e JTOpen : offre maggiori informazioni su ToolboxME per iSeries.
- IBM alphaWork Wireless : offre informazioni su nuove tecnologie wireless, compresi gli scaricamenti e i collegamenti alle risorse di sviluppo.

- Sun Java 2 Platform, Micro Edition : fornisce informazioni aggiuntive sulle tecnologie Java wireless, inclusi:
  - KVM (K Virtual Machine)
  - CLDC (Connected Limited Device Configuration)
  - MIDP (Mobile Information Device Profile)
- Java Wireless Developer : offre un'ampia gamma di informazioni tecniche per gli sviluppatori di applicazioni wireless Java.
- Strumenti di sviluppo dell'applicazione wireless:
  - IBM WebSphere Studio Device Developer
  - Java 2 Platform Micro Edition, Wireless Toolkit

Java

Java è un linguaggio di programmazione che consente di sviluppare le applicazioni portabili orientate all'oggetto e le applet. Utilizzare i seguenti siti per acquisire ulteriori informazioni su Java:

- IBM developerWorks Java technology zone : offre le informazioni, l'addestramento e gli strumenti di supporto all'utilizzo di Java, dei prodotti IBM e di altre tecnologie al fine di creare soluzioni aziendali.
- IBM alphaWorks Java : offre informazioni sulle nuove tecnologie Java, inclusi gli scaricamenti e i collegamenti alle risorse di sviluppo.
- "The Source for Java Technology" from Sun Microsystems : offre informazioni sui vari utilizzi di Java, comprese le nuove tecnologie.
- Foundation - Java, IBM eServer enablement Technical resources : fornisce informazioni su Java e sulla modalità di utilizzo possibile ed effettiva da parte dei partner aziendali IBM.

Denominazione Java e interfaccia indirizzario

- Java Naming and Directory Interface(TM) (JNDI) : offre una panoramica su JNDI, informazioni tecniche, esempi e un elenco dei tecnici della manutenzione disponibili.
- iSeries Directory Server (LDAP) : fornisce informazioni su LDAP (Lightweight Directory Access Protocol) su OS/400.

Java Secure Socket Extension

- Java Secure Socket Extension (JSSE) : offre una breve panoramica di JSSE e collegamenti ad ulteriori fonti di informazioni.

Servlet

I servlet sono piccoli programmi Java eseguiti su un server e che rimangono come intermediari delle richieste da uno o più client (ognuno dei quali viene eseguito su un browser) ad uno o più database. Poiché i servlet sono programmati in Java, possono eseguire richieste come sottoprocessi multipli all'interno di una singola elaborazione, risparmiando in questo modo le risorse del sistema. Utilizzare i seguenti siti per ulteriori informazioni sui servlet:

- IBM Websphere, IBM PartnerWorld : offre informazioni sul server dell'applicazione Web basato sul servlet.

- Java Servlet technology : fornisce informazioni tecniche, istruzioni e strumenti per conoscere ed utilizzare i servlet.

XHTML

Si considera XHTML il successore di HTML 4.0. Si basa su HTML 4.0, ma incorpora l'estendibilità di XML. Utilizzare i seguenti siti per ulteriori informazioni su XHTML:

- The Web Developer's Virtual Library : offre un'introduzione a XHTML, compresi gli esempi ed i collegamenti per ulteriori informazioni.
- W3C : fornisce informazioni tecniche sugli standard e le raccomandazioni XHTML.

XML

XML (Extensible Markup Language) è un metalinguaggio che consente di descrivere e organizzare le informazioni secondo modalità facilmente comprensibili ai computer e alle persone. Un metalinguaggio consente di definire un document markup language e la sua struttura. Utilizzare i seguenti siti per ulteriori informazioni su XML:

- IBM developerWorks XML zone : fornisce un sito dedicato all'uso che IBM fa con di XML e al modo in cui facilita l'e-commerce.
- IBM alphaWorks XML : offre informazioni sugli standard e sugli strumenti XML emergenti, compresi gli scaricamenti e i collegamenti alle risorse di sviluppo.
- Foundation - XML, IBM eServer enablement Technical resources : fornisce informazioni sull'XML e sulle modalità di utilizzo possibili ed effettive da parte dei partner aziendali IBM .
- W3C XML : offre risorse tecniche per gli sviluppatori XML.
- XML.com : offre informazioni aggiornate su XML nelle aziende di computer
- XML.org : fornisce novità e informazioni sulla comunità XML comprese le novità del settore di produzione, i calendari degli eventi ed altro.
- XML Cover Pages : fornisce un lavoro di riferimento in linea esaustivo per XML, SGML per gli standard XML correlati, come XSL e XSLT.

Altri riferimenti

- IBM HTTP Server for iSeries : fornisce informazioni, risorse e suggerimenti sul IBM HTTP Server for iSeries.
- iSeries Access for Windows : offre informazioni su iSeries Access for Windows, inclusi gli scaricamenti, le FAQ e i collegamenti ad ulteriori siti.
- IBM WebSphere Host On-Demand : fornisce informazioni sull'emulazione basata sul browser che offre il supporto per S/390, iSeries e per l'emulazione DEC/Unix.
- IBM Support and downloads : offre un portale al supporto hardware e software di IBM.

Informazioni sull'Esonero di responsabilità per gli esempi di codice

Questo documento contiene esempi di codice da utilizzare per le esigenze di programmazione.

L'IBM fornisce una licenza non esclusiva per utilizzare tutti gli esempi del codice di programmazione da cui creare funzioni simili personalizzate, in base a richieste specifiche.

Questo codice di esempio è fornito dall'IBM con la sola funzione illustrativa. Questi esempi non sono stati interamente testati in tutte le condizioni. IBM, perciò, non fornisce nessun tipo di garanzia o affidabilità implicita, rispetto alla funzionalità o alle funzioni di questi programmi.

Tutti i programmi qui contenuti vengono forniti all'utente "COSI' COME SONO" senza garanzie di alcun tipo. Le garanzie implicite di non contraffazione, commerciabilità e adeguatezza a scopi specifici sono espressamente vietate.

Disposizioni per il download e la stampa delle pubblicazioni

Le autorizzazioni per l'utilizzo delle pubblicazioni da scaricare vengono concesse in base alle seguenti disposizioni ed alla loro accettazione.

Uso personale: E' possibile riprodurre queste Pubblicazioni per uso personale, non commerciale a condizione che vengano conservate tutte le indicazioni relative alla proprietà.Non è possibile distribuire, visualizzare o produrre lavori derivati di tali Pubblicazioni o di qualsiasi loro parte senza chiaro consenso da parte di IBM.

Uso commerciale: E' possibile riprodurre, distribuire e visualizzare queste Pubblicazioni unicamente all'interno del proprio gruppo aziendale a condizione che vengano conservate tutte le indicazioni relative alla proprietà.Non è possibile effettuare lavori derivati di queste Pubblicazioni o riprodurre, distribuire o visualizzare queste Pubblicazioni o qualsiasi loro parte al di fuori del proprio gruppo aziendale senza chiaro consenso da parte di IBM.

Fatto salvo quanto espressamente concesso in questa autorizzazione, non sono concesse altre autorizzazioni, licenze o diritti, espressi o impliciti, relativi alle Pubblicazioni o a qualsiasi informazione, dato, software o altra proprietà intellettuale qui contenuta.

IBM si riserva il diritto di ritirare le autorizzazioni qui concesse qualora, a propria discrezione, l'utilizzo di queste Pubblicazioni sia a danno dei propri interessi o, come determinato da IBM, qualora non siano rispettate in modo appropriato le suddetta istruzioni.

Non è possibile scaricare, esportare o ri-esportare queste informazioni se non pienamente conformi con tutte le leggi e le norme applicabili, incluse le leggi e le norme di esportazione degli Stati Uniti. IBM NON RILASCIA ALCUNA GARANZIA RELATIVAMENTE AL CONTENUTO DI QUESTE PUBBLICAZIONI. LE PUBBLICAZIONI SONO FORNITE "NELLO STATO IN CUI DI TROVANO" SENZA ALCUN TIPO DI GARANZIA, ESPRESSA O IMPLICITA, INCLUSE, A TITOLO ESEMPLIFICATIVO, GARANZIE IMPLICITE DI COMMERCIABILITA' ED IDONEITA' PER UNO SCOPO PARTICOLARE.

Tutto il materiale è tutelato dal copyright da IBM Corporation.

Con il download o la stampa di una pubblicazione da questo sito, si accettano queste disposizioni.