

- [IBM Toolbox per Java](#)
  - [Novità in V5R2](#)
  - [Stampare questo argomento](#)
  - [Informazioni preliminari](#)
    - [Gestire l'installazione](#)
    - [Installare Toolbox per Java](#)
      - [Requisiti OS/400](#)
        - [Opzioni OS/400 necessarie](#)
          - [Determinare se Toolbox per Java è installato](#)
          - [Verificare il profilo utente QUSER](#)
          - [Modificare il profilo utente QUSER](#)
        - [Dipendenze su altri programmi](#)
        - [Compatibilità OS/400](#)
        - [Ottimizzazioni native](#)
        - [Requisiti ToolboxME per iSeries](#)
      - [Requisiti stazione di lavoro](#)
        - [Eeguire le applicazioni Java](#)
        - [Eeguire le applet Java](#)
        - [Requisiti ToolboxME per iSeries](#)
        - [Requisiti Swing](#)
      - [Installare sul server iSeries](#)
      - [Installare sulla stazione di lavoro](#)
        - [File jar](#)
    - [Proprietà di sistema](#)
    - [Esempi semplici](#)
  - [Classi](#)
    - [Classi Access](#)
      - [Classe AS400](#)
      - [Classe SecureAS400](#)
      - [AS400JPing](#)
      - [Classe BidiTransform](#)
      - [Classi ClusteredHashTable](#)
      - [Classe CommandCall](#)
      - [Classe ConnectionPool](#)
      - [Classe DataArea](#)
      - [Classi DataConversion e DataDescription](#)
        - [Classi di conversione numerica](#)
        - [Classi di conversione testo \(carattere\)](#)

- [Classi di conversione composita \(numerico e testo\)](#)
- [Classi FieldDescription](#)
- [Classe RecordFormat](#)
- [Classe Record](#)
- [Classe LineDataRecordWriter](#)
- [Classi DataQueue](#)
  - [Code dati sequenziali](#)
  - [Code dati con chiave](#)
- [Classi di certificato digitale](#)
- [Classe EnvironmentVariable](#)
- [Eccezioni](#)
- [Classi FTP](#)
- [Classi IFS \(Integrated file system\)](#)
  - [Classe IFSFile](#)
  - [Classe IFSJavaFile](#)
  - [Classe IFSFileInputStream](#)
  - [Classe IFSTextFileInputStream](#)
  - [Classe IFSFileOutputStream](#)
  - [Classe IFSTextFileOutputStream](#)
  - [Classe IFSRandomAccessFile](#)
  - [Classe IFSFileDialog](#)
  - [Classe IFSKey](#)
  - [Metodi di condivisione](#)
- [Classe JavaApplicationCall](#)
- [Classi JDBC](#)
  - [Potenziamenti JDBC](#)
  - [Proprietà JDBC](#)
  - [Classe Blob](#)
  - [Classe CallableStatement](#)
  - [Classe Clob](#)
  - [Classe Connection](#)
  - [Classe ConnectionPool](#)
  - [Classe DatabaseMetaData](#)
  - [Classe DataSource](#)
  - [Classe Driver](#)
  - [Classe ParameterMetaData](#)
  - [Classe PreparedStatement](#)
  - [Classi ResultSet e ResultSetMetaData](#)

- [Classe RowSet](#)
- [Classe Savepoint](#)
- [Classe Statement](#)
- [Classe XAConnection e XAResource](#)
- [Classi Jobs](#)
  - [Classe Job](#)
  - [Classe JobList](#)
  - [Classe JobLog](#)
- [Classe AS400Message](#)
- [Classe NetServer](#)
- [Classi Permission e UserPermission](#)
  - [Classe DLOPermission](#)
  - [Classe QSYSPermission](#)
  - [Classe RootPermission](#)
- [Classi Print](#)
  - [Classe PrintObjectList](#)
  - [Classe PrintObject](#)
    - [Richiamare attributi PrintObject](#)
    - [Attributi file di stampa](#)
    - [Attributi file di spool](#)
  - [Classe SpooledFileOutputStream](#)
  - [Classi SCSWriter](#)
  - [Classe PrintObjectInputStream](#)
  - [Classi PrintObjectPageInputStream e PrintObjectTransformedInputStream](#)
- [Classe ProductLicense](#)
- [Classe ProgramCall](#)
- [Classe QSYSObjectPathName](#)
- [Classi di accesso al livello record](#)
  - [Classe AS400File](#)
  - [Classe KeyedFile](#)
  - [Classe SequentialFile](#)
  - [Classe AS400FileRecordDescription](#)
- [Classe ServiceProgramCall](#)
- [Classe SystemStatus](#)
- [Classi SystemValue](#)
- [Classe Trace](#)
- [Classi UserGroup e UserList](#)
- [Classe UserSpace](#)

- [Classi HTML](#)
  - [Classe BidiOrdering](#)
  - [Classe HTMLAlign](#)
  - [Classi HTML Form](#)
    - [Classi Form Input](#)
      - [ButtonFormInput](#)
      - [FileFormInput](#)
      - [HiddenFormInput](#)
      - [ImageFormInput](#)
      - [ResetFormInput](#)
      - [SubmitFormInput](#)
      - [TextFormInput](#)
      - [PasswordFormInput](#)
      - [RadioFormInput](#)
      - [CheckboxFormInput](#)
    - [Classi LayoutFormPanel](#)
      - [GridLayoutFormPanel](#)
      - [LinearLayoutFormPanel](#)
    - [Classe TextAreaFormElement](#)
    - [Classe LabelFormElement](#)
    - [Classe SelectFormElement](#)
    - [Classe SelectOption](#)
    - [Classe RadioFormInputGroup](#)
  - [Classe HTMLHeading](#)
  - [Classe HTMLHyperlink](#)
  - [HTMLImage](#)
  - [Classi HTMLList](#)
  - [Classe HTMLMeta](#)
  - [Classe HTMLParameter](#)
  - [Classe HTMLServlet](#)
  - [Classi HTML Table](#)
    - [Classe HTMLTableCell](#)
    - [Classe HTMLTableRow](#)
    - [Classe HTMLTableHeader](#)
    - [Classe HTMLTableCaption](#)
  - [Classe HTMLText](#)
  - [Classi HTMLTree](#)
    - [Classe HTMLTreeElement](#)

- [Classe FileTreeElement](#)
- [Classe FileListElement](#)
- [Classe FileListRenderer](#)
- [Classi ReportWriter](#)
  - [Classi Context](#)
  - [Classe JSPReportProcessor](#)
  - [Classe XSLReportProcessor](#)
- [Classi Resource](#)
  - [Classe Resource](#)
  - [Classe ResourceList](#)
  - [Classe Presentation](#)
- [Classi di sicurezza](#)
  - [SSL \(Secure Sockets Layer\)](#)
    - [Responsabilità legali SSL](#)
    - [Utilizzare SSL sui server iSeries](#)
      - [Impostare server iSeries per utilizzare SSL](#)
      - [Utilizzare certificati da autorità garantite](#)
      - [Utilizzare autocertificazioni](#)
    - [Utilizzare SSL su server proxy](#)
      - [Impostare server proxy per utilizzare SSL](#)
      - [Impostare client proxy per utilizzare SSL](#)
  - [Servizi di autenticazione](#)
- [Classi Servlet](#)
  - [Classi Authentication](#)
  - [Classe RowData](#)
    - [Classe ListRowData](#)
    - [Classe RecordListRowData](#)
    - [Classe ResourceListRowData](#)
    - [Classe SQLResultSetRowData](#)
  - [Classe RowMetaData](#)
    - [Classe ListMetaData](#)
    - [Classe RecordFormatMetaData](#)
    - [Classe SQLResultSetMetaData](#)
  - [Classi Converter](#)
    - [Classe StringConverter](#)
    - [Classe HTMLFormConverter](#)
    - [Classe HTMLTableConverter](#)
- [Classi programma di utilità](#)

- [Classe AS400ToolboxInstaller](#)
- [Classe AS400ToolboxJarMaker](#)
  - [Componenti supportati](#)
  - [CCSID e valori di codifica supportati](#)
- [Classe CommandPrompter](#)
- [Classi RunJavaApplication e VRunJavaApplication](#)
- [Classe JPing](#)
- [Classi Vaccess](#)
  - [Diagramma delle classi del componente GUI](#)
  - [Classi AS400Pane](#)
  - [Classi CommandCall](#)
  - [Classi DataQueue](#)
  - [Eventi errore](#)
  - [Classi IFS](#)
    - [Classe VIFSFileDialog](#)
    - [Classe VIFSDirectory](#)
    - [Classe IFSTextFileDocument](#)
  - [Classe VJavaApplicationCall](#)
  - [Classi JDBC \(SQL\)](#)
    - [Classi SQLStatementButton e SQLStatementMenuItem](#)
    - [Classe SQLStatementDocument](#)
    - [Classe SQLResultSetFormPane](#)
    - [Classe SQLResultSetTablePane](#)
    - [Classe SQLResultSetTableModel](#)
    - [Classe SQLQueryBuilderPane](#)
  - [Classi VJobList e VJob](#)
  - [Classi Message](#)
    - [Classe VMessageList](#)
    - [Classe VMessageQueue](#)
  - [Utilizzare le informazioni della classe Permission](#)
  - [Classi Print](#)
    - [Classe VPrinters](#)
    - [Classe VPrinter](#)
    - [Classe VPrinterOutput](#)
    - [Classe SpooledFileViewer](#)
  - [Classi ProgramCall e ProgramParameter](#)
  - [Classi di accesso al livello record](#)
    - [Classe RecordListFormPane](#)

- [Classe RecordListTablePane](#)
    - [Classe RecordListTableModel](#)
  - [Classi ResourceList](#)
  - [Classi SystemStatus](#)
    - [Classe VSystemStatusPane](#)
  - [Classe VSystemValue](#)
  - [Classi Users e Groups](#)
- [Graphical Toolbox](#)
  - [Installare il Graphical Toolbox](#)
  - [Creare l'interfaccia utente](#)
  - [Visualizzare i pannelli al tempo di esecuzione](#)
  - [Creare file di aiuto in linea](#)
  - [Esempi di Graphical Toolbox](#)
  - [Utilizzare il Graphical Toolbox in un browser](#)
  - [Barra degli strumenti del programma di creazione del pannello](#)
- [JavaBean](#)
- [JDBC](#)
- [PCML \(Program Call Markup Language\)](#)
  - [Elaborazione PCML](#)
  - [Sintassi PCML](#)
    - [Tag Program](#)
    - [Tag Struct](#)
    - [Tag Data](#)
    - [Valori per length e precision](#)
- [Supporto proxy](#)
- [RFML \(Record Format Markup Language\)](#)
  - [Requisiti](#)
  - [Esempio RFML](#)
    - [Esempio: file sorgente RFML](#)
  - [Classe RecordFormatDocument](#)
  - [Documenti e sintassi RFML](#)
    - [DTD RFML](#)
    - [Tag Data](#)
    - [Tag Rfml](#)
    - [Tag Recordformat](#)
    - [Tag Struct](#)
- [Sicurezza](#)
- [Programma di debug del sistema iSeries](#)

- [Componenti](#)
- [Installazione](#)
- [Eseguire il programma di debug del sistema iSeries](#)
- [Proprietà di sistema](#)
  - [Esempio: file delle proprietà](#)
  - [Esempio: file sorgente classe proprietà di sistema](#)
- [ToolboxME per iSeries](#)
  - [Requisiti](#)
  - [Scaricare ed installare](#)
  - [Concetti](#)
  - [Classi](#)
    - [Classe MEServer](#)
    - [Classe AS400](#)
    - [Classe CommandCall](#)
    - [Classe DataQueue](#)
    - [Classe ProgramCall](#)
    - [Classe JdbcMe](#)
      - [JdbcMeConnection](#)
      - [JdbcMeDriver](#)
      - [JdbcMeLiveResultSet](#)
      - [JdbcMeOfflineData](#)
      - [JdbcMeOfflineResultSet e JdbcMeResultSetMetaData](#)
      - [JdbcMeStatement](#)
  - [Creare un'applicazione](#)
  - [Esempi](#)
- [FAQ \(Frequently asked questions\)](#)
- [Esempi](#)
  - [Classi Access](#)
    - [Esempio: utilizzare CommandCall](#)
    - [Esempio: utilizzare ConnectionPool](#)
    - [Esempio: utilizzare classi DataQueue \(con Record e RecordFormat\) per inserire i dati in una coda](#)
    - [Esempio: utilizzare classi DataQueue \(con Record e RecordFormat\) per leggere dati da una coda](#)
    - [Esempi: utilizzare IFSFile](#)
    - [Esempio: utilizzare il metodo IFSFile.listFiles\(\)](#)
    - [Esempio: utilizzare le classi IFSFile per copiare i file](#)
    - [Esempio: utilizzare le classi IFSFile per elencare il contenuto di un indirizzario](#)



- [Esempio: utilizzare le classi JDBC per creare e popolare una tabella](#)
- [Esempio: utilizzare le classi JDBC per interrogare la tabella](#)
- [Esempio: utilizzare JobList per elencare informazioni sull'ID lavoro](#)
- [Esempio: utilizzare JobList per richiamare una lista di lavori](#)
- [Esempio: utilizzare JobLog](#)
- [Esempio: utilizzare le classi Print per creare file di spool](#)
- [Esempio: utilizzare le classi Print per creare file di spool SCS](#)
- [Esempio: utilizzare le classi Print per leggere file di spool](#)
- [Esempio: utilizzare le classi Print per elencare in modo asincrono i file di spool \(utilizzando i listener\)](#)
- [Esempio: utilizzare le classi Print per elencare in modo asincrono i file di spool \(senza utilizzare i listener\)](#)
- [Esempio: utilizzare le classi Print per elencare in modo sincrono i file di spool](#)
- [Esempio: utilizzare ProgramCall per richiamare lo stato del sistema](#)
- [Esempio: utilizzare le classi di accesso a livello record per accedere ad un file](#)
- [Esempio: utilizzare le classi di accesso al livello record per leggere un file](#)
- [Esempio: utilizzare le classi di accesso al livello record per leggere i record tramite chiave](#)
- [Esempio: utilizzare UserList per elencare tutti gli utenti in un gruppo](#)
- [Bean](#)
  - [Esempio: utilizzare i listener per stampare i commenti](#)
  - [Esempio: creare pulsanti che eseguono i comandi](#)
- [Graphical Toolbox](#)
  - [Esempio: come costruire e visualizzare un pannello](#)
  - [Esempio: gestire le caselle combinate modificabili](#)
  - [Esempio: utilizzare il GUI Builder per creare pannelli](#)
  - [Esempio: utilizzare il GUI Builder per creare pannelli sovrapposti](#)
  - [Esempio: utilizzare il GUI Builder per creare fogli delle proprietà](#)
  - [Esempio: utilizzare il GUI Builder per creare pannelli suddivisi](#)
  - [Esempio: utilizzare il GUI Builder per creare pannelli con separatori](#)
  - [Esempio: utilizzare il GUI Builder per creare wizard](#)
  - [Esempio: utilizzare il GUI Builder per creare barre degli strumenti](#)
  - [Esempio: utilizzare il GUI Builder per creare barre dei menu](#)
  - [Esempio: utilizzare il GUI Builder per creare documenti di aiuto](#)
  - [Esempio: modificare i documenti di aiuto creati dal GUI Builder](#)
  - [Esempio: esaminare un programma PDML](#)
- [Classi HTML](#)
  - [Esempio: utilizzare le classi HTMLForm](#)
  - [Esempio: utilizzare la classe HTMLTree](#)

- [Esempio: creare una gerarchia ad albero IFS \(integrated file system\) \(parte 1 di 3\)](#)
- [Esempio: creare una gerarchia ad albero IFS \(parte 2 di 3\)](#)
- [Esempio: creare una gerarchia ad albero IFS \(parte 3 di 3\)](#)
- [Esempio: utilizzare le classi HTMLTable](#)
- [PCML](#)
  - [Esempio: richiamare dati](#)
  - [Esempio: richiamare una lista di informazioni](#)
  - [Esempio: richiamare dati multidimensionali](#)
- [Classi ReportWriter](#)
  - [Esempio: utilizzare JSPReportProcessor con PDFContext](#)
  - [Esempio: file JSP di esempio JSPReportProcessor](#)
  - [Esempio: utilizzare XSLReportProcessor con PCLContext](#)
  - [Esempio: file XML di esempio XSLReportProcessor](#)
  - [Esempio: file XML di esempio XSLReportProcessor](#)
- [Classi Resource](#)
  - [Esempio: richiamare un valore di attributo da RUser](#)
  - [Esempio: impostare valori di attributo per RJob](#)
  - [Esempio: utilizzare un codice generico per accedere alle risorse](#)
  - [Esempi: utilizzare ResourceList](#)
- [RFML](#)
- [Classi di sicurezza](#)
- [Classi Servlet](#)
  - [Esempio: utilizzare la classe ListRowData](#)
  - [Esempio: utilizzare la classe RecordListRowData](#)
  - [Esempio: utilizzare la classe ResultSetRowData](#)
  - [Esempio: utilizzare la classe HTMLFormConverter](#)
  - [Esempio: utilizzare le classi servlet e HTML insieme](#)
- [Esempi semplici](#)
  - [Scrivere il primo programma Toolbox per Java](#)
  - [Chiamare i comandi](#)
  - [Utilizzare le code di messaggi \(parte 1 di 3\)](#)
  - [Utilizzare le code di messaggi \(parte 2 di 3\)](#)
  - [Utilizzare le code di messaggi \(parte 3 di 3\)](#)
  - [Utilizzare l'accesso al livello record \(parte 1 di 2\)](#)
  - [Utilizzare l'accesso al livello record \(parte 2 di 2\)](#)
  - [Utilizzare le classi JDBC per creare una tabella \(parte 1 di 2\)](#)
  - [Utilizzare le classi JDBC per creare una tabella \(parte 2 di 2\)](#)
  - [Visualizzare una lista di lavori server in una GUI](#)

- [Suggerimenti per la programmazione](#)
- [ToolboxME per iSeries](#)
  - [Esempio: utilizzare ToolboxME per iSeries, MIDP e JDBC](#)
  - [Esempio: utilizzare ToolboxME per iSeries, MIDP e Toolbox per Java](#)
- [Classi programma di utilità](#)
  - [Esempio: utilizzare AS400ToolboxInstaller per installare Toolbox per Java](#)
  - [Esempio: utilizzare CommandPrompter per richiedere ed eseguire un comando](#)
- [Classi Vaccess](#)
  - [Esempio: utilizzare AS400ListPane](#)
  - [Esempio: utilizzare AS400DetailsPane](#)
  - [Esempio: utilizzare AS400TreePane](#)
  - [Esempio: utilizzare AS400ExplorerPane](#)
  - [Esempio: utilizzare CommandCallMenuItem](#)
  - [Esempio: utilizzare DataQueueDocument](#)
  - [Esempio: creare AS400JDBCDataSourcePane](#)
  - [Esempio: utilizzare VJobList per presentare una lista di lavori](#)
  - [Esempio: utilizzare ProgramCallButton](#)
- [Suggerimenti per la programmazione](#)
  - [Chiudere il programma Java](#)
  - [Nomi percorso IFS \(Integrated File System\)](#)
  - [Gestire i collegamenti](#)
  - [JVM OS/400](#)
    - [Effettuare un confronto tra JVM OS/400 e le classi IBM Toolbox per Java](#)
    - [Eseguire le classi](#)
    - [Impostare il nome sistema, l'ID utente e la parola d'ordine](#)
  - [IASP \(Independent auxiliary storage pool\)](#)
  - [Ottimizzazione OS/400](#)
  - [Miglioramenti delle prestazioni](#)
  - [NLS \(National language support\)](#)
  - [Servizio e supporto](#)
- [Informazioni correlate](#)
- [Esonero di responsabilità per il codice](#)

# IBM Toolbox per Java

IBM Toolbox per Java rappresenta una serie di classi Java<sup>(TM)</sup> che consentono di utilizzare i programmi Java per accedere ai dati sui server iSeries e AS/400e. E' possibile utilizzare queste classi per scrivere le applicazioni client/server, le applet ed i servlet che gestiscono i dati su iSeries. E' possibile anche eseguire le applicazioni Java che utilizzano le classi di IBM Toolbox per Java sulla JVM (Java virtual machine) di iSeries.

IBM Toolbox per Java utilizza i [Server host](#) di iSeries come punti di accesso al sistema. Siccome Toolbox per Java utilizza le funzioni di comunicazione incorporate in Java, non è necessario utilizzare IBM iSeries Access Express per Windows per utilizzare 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.

Per ulteriori informazioni su IBM Toolbox per Java utilizzare la barra di navigazione principale o i seguenti collegamenti:

## [Novità in V5R2](#)

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

## [Stampare l'argomento IBM Toolbox per Java](#)

Visualizzare o scaricare un PDF dell'argomento Toolbox per Java. E' possibile anche scaricare l'argomento Toolbox per Java in un pacchetto compresso.

## [Utilizzare il rilevatore classe](#)

Eseguire una ricerca facile e veloce delle classi per nome e descrizione, visualizzare le classi per pacchetto oppure esaminare una lista in ordine alfabetico di tutte le classi di Toolbox per Java.

## [Informazioni preliminari di IBM Toolbox per Java](#)

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

## [Classi IBM Toolbox per Java](#)

Leggere le informazioni sulle varie classi nei pacchetti IBM Toolbox per Java che consentono di gestire i dati del server iSeries e AS/400e. Queste informazioni includono spiegazioni, codici di esempio ed informazioni tecniche che facilitano la creazione di programmi IBM Toolbox per Java.

## [Utilizzare 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 di iSeries Navigator.

## [JavaBean](#)

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

## [JDBC](#)

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

## [Utilizzare PCML per richiamare programmi iSeries](#)

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, basata su XML, che descrive completamente i parametri immissione ed emissione per i programmi iSeries

richiamati dall'applicazione Java.

### [Supporto proxy](#)

Acquisire informazioni su come utilizzare il supporto proxy di IBM Toolbox per Java, che include l'utilizzo del protocollo SSL (Sockets Layer) per codificare i dati.

### [»Utilizzare RFML per definire e gestire i formati di dati](#)

Utilizzare RFML (Record Format Markup Language) per separare le specifiche del formato dati dalla logica aziendale dei programmi Java. RFML è una sintassi di tag, basata su XML e strettamente correlata a PCML, che consente alle applicazioni Java di specificare e gestire i campi all'interno di determinati tipi di record.◀

### [»Sicurezza](#)

Leggere informazioni sull'utilizzo di JSSE (Java Secure Socket Extension) e Toolbox per Java per fornire uno scambio sicuro di dati tra client e server che eseguono un qualsiasi protocollo di applicazione su TCP/IP.◀

### [»Programma di debug del sistema iSeries](#)

Utilizzare la GUI (graphical user interface) del programma di debug del sistema iSeries per effettuare il debug e una verifica dei programmi in esecuzione sul server iSeries.◀

### [Proprietà di sistema](#)

Acquisire informazioni sull'utilizzo delle proprietà di sistema per configurare diversi aspetti di IBM Toolbox per Java, ad esempio, quando definire 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 di Toolbox per Java per scrivere programmi Java per varie unità senza fili. Utilizzando ToolboxME per iSeries, le unità senza fili possono accedere direttamente ai dati e alle risorse del server iSeries.◀

### [Javadoc relativi a IBM Toolbox per Java](#)

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

### [»FAQ \(Frequently asked question\)](#)

Reperire le risposte alle domande sull'ottimizzazione delle prestazioni di IBM Toolbox per Java, eseguendo la risoluzione dei problemi, utilizzando JDBC ed altro ancora.◀


Ulteriori informazioni includono:

- [Una lista di esempi di programmazione di Toolbox per Java](#)
- [Suggerimenti di programmazione](#) per facilitare l'utilizzo di 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 V5R2

IBM Toolbox per Java è disponibile nei seguenti formati:

- Il programma su licenza per IBM Toolbox per Java, 5722-JC1, Versione 5 Rilascio 2 (V5R2) viene installato su V4R5 o nelle successive versioni di OS/400. Da un client, IBM Toolbox per Java si ricollega a V4R5 e a 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](#).
- Ora IBM Toolbox per Java è anche un sorgente aperto. E' possibile scaricare il codice e ricevere ulteriori informazioni dal sito web [JTOpen](#) .

## Nuovi pacchetti

[IBM Toolbox for Java MicroEdition](#) è un nuovo pacchetto per IBM Toolbox per Java. Il nuovo pacchetto [com.ibm.as400.micro](#) fornisce una serie di classi che consentono all'utente di scrivere programmi Java in esecuzione su unità senza fili come i PDA (Personal Digital Assistant) e i cellulari. E' necessario [scaricare separatamente](#) il pacchetto micro.

Il [programma di debug di sistema iSeries](#) fornisce alla stazione di lavoro un nuovo ambiente grafico di debug per i propri programmi ILE, Java, C e C++ in esecuzione su un server iSeries.

## Nuove classi

Inoltre, V5R2 di IBM Toolbox per Java presenta molte classi nuove nei pacchetti esistenti. Le nuove classi consentono di:

- Utilizzare le classi [ClusteredHashTable](#) per dividere e replicare i dati non persistenti tra i nodi in un cluster
- Utilizzare la classe [CommandPrompter](#) per richiedere i parametri su un comando dato.
- Per le nuove classi JDBC<sup>(TM)</sup>, consultare [Le nuove classi JDBC e le funzioni avanzate..](#)
- La classe [RecordFormatDocument](#) consente di utilizzare il nuovo componente RFML (Record Format Markup Language) per specificare i formati record e per creare, leggere e scrivere i record dei dati.

## Classi potenziate

V5R2 di IBM Toolbox per Java include anche gli aggiornamenti alle classi esistenti. Gli aggiornamenti offrono:

- L'aggiunta del supporto Kerberos agli oggetti [AS400](#), che ora utilizzano la framework JGSS (Java Generic Security Service) per effettuare autenticazioni nei server di Toolbox per Java.
- Gli [oggetti SecureAS400](#) possono ora utilizzare la framework JSSE (Java Secure Socket Extention) per codificare il flusso di dati tra il client e il server.
- Modifiche ad alcune tag PCML (Program Call Markup Language) e funzionalità:
  - I nuovi attributi per la tag [<data>](#) aggiungono il supporto per le stringhe Unicode e consentono agli utenti di specificare come ridimensionare gli spazi dalle stringhe.
  - Gli attributi nuovi e modificati per la [tag <program>](#) consentono ora di specificare il percorso al momento dell'esecuzione e consentono una specifica CCSID per il nome del punto di immissione del programma di servizio.
  - Per effettuare una traccia, PCML richiede ora l'utilizzo della classe Trace

([com.ibm.as400.access.Trace](#)) invece della classe PcmlMessageLog.

## Nuove classi JDBC e funzioni potenziate

Il supporto JDBC V5R2 di IBM Toolbox per Java presenta nuove classi e funzioni potenziate, incluso il supporto per l'API (application programming interface) JDBC 3.0. Le Modifiche significative comprendono le seguenti:

- [AS400JDBCsavepoint](#) è una nuova classe (che supporta JDBC 3.0) che fornisce un controllo più accurato sui rollback della transazione
- [AS400JDBCParameterMetaData](#) è una nuova classe (che supporta JDBC 3.0) che consente di richiamare i tipi e le proprietà dei parametri negli oggetti PreparedStatement e CallableStatement
- Ulteriore capacità di collegamento a [IASP \(independent auxiliary storage pools\)](#)
- Nuovi metodi in [blob \(binary large object\)](#) e [clob \(character large object\)](#) (che supportano JDBC 3.0) consentono di aggiornare i valori di questi tipi di dati
- Una nuova proprietà JDBC ([metadati estesi](#)) migliora il prospetto degli attributi [ResultSetMetaData](#)
- [Altri aggiornamenti che migliorano il supporto JDBC](#)

## Nuovo componente XML

Nella V5R2 dell'IBM Toolbox per Java è stato aggiunto l'[RFML \(Record Format Markup Language\)](#), una estensione di XML simile a PCML. RFML consente di utilizzare l'XML nei programmi Java per specificare il formato dei buffer dei dati e i formati record file fisico e di specificare il contenuto dei buffer e dei record richiamati.


## Funzioni e caratteristiche ulteriori in Graphical Toolbox

[Graphical Toolbox](#) incorpora nuove funzioni:

- Visualizzare le celle nella colonna della tabella come caselle di spunta
- Specificare le dimensioni minime (di altezza ed ampiezza) per le finestre di dialogo per assicurare una visualizzazione adeguata degli elementi della finestra di dialogo
- Specificare che la prima colonna nella tabella contiene una gerarchia ad albero dinamica, in cui ogni cella corrisponde a un nodo dell'albero

Per ulteriori informazioni su queste funzioni, consultare l'aiuto in linea GUI Builder.

## Compatibilità

IBM Toolbox per Java non supporta più l'esecuzione nella JVM predefinita in Netscape<sup>(R)</sup> Navigator o Microsoft<sup>(R)</sup> Internet Explorer. Per fare in modo che la propria applet che utilizza le classi Toolbox per Java venga eseguita in un browser, è necessario installare il [Modulo aggiuntivo JRE \(Java2 Runtime Environment\) 1.3 o di Sun](#) .

Toolbox per Java non contiene 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 rilascio di IBM Toolbox per Java per deserializzare alcuni oggetti serializzati con

rilasci precedenti alla 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 rilascio di IBM Toolbox per Java non funziona con la V4R5 e con le versioni precedenti di iSeries Client Encryption.

IBM Toolbox per Java continua a fornire supporto per

- Swing 1.1, necessario per utilizzare le classi GUI o il Graphical Toolbox
- Java 2 Platform, Standard Edition (J2SE), con supporto continuo di Java Development Kit 1.1.8

E' necessario inoltre esaminare [Requisiti OS/400 per l'esecuzione di IBM Toolbox per Java](#).



## Novità a partire dal 26 settembre 2002


### [Rilevatore classe IBM Toolbox per Java](#)

Utilizzare il nuovo rilevatore classe per eseguire una ricerca facile e veloce delle classi per nome e descrizione, visualizzare le classi per pacchetto oppure esaminare una lista in ordine alfabetico di tutte le classi di Toolbox per Java. La breve descrizione di ogni classe consente un collegamento a informazioni più specifiche.

## Come riconoscere novità e modifiche

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

-  indica l'inizio delle informazioni nuove o modificate
-  indica la fine delle informazioni nuove o modificate

Per ulteriori informazioni sulle novità o sulle modifiche di questo rilascio, consultare [Memo per gli utenti](#) .



# Stampare questo argomento

Per visualizzare o scaricare la versione PDF, selezionare [PDF IBM Toolbox per Java](#) (circa 5.4 megabyte o 811 pagine).


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

## Salvare i file PDF

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

1. Fare clic con il tastino destro del mouse sul PDF nel browser (fare clic con il tastino destro del mouse sul collegamento riportato sopra).
2. Fare clic su **Salva destinazione con nome**
3. Andare all'indirizzario nel quale si desidera salvare il PDF.
4. Fare clic su **Salva**.

## Scaricare Adobe Acrobat Reader

Se è necessario Adobe Acrobat Reader per visualizzare o stampare questi PDF, è possibile scaricarlo una copia dal [sito Web Adobe](http://www.adobe.com/products/acrobat/readstep.html) (www.adobe.com/products/acrobat/readstep.html) .

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

1. Aprire il PDF sul browser (fare clic sul collegamento sopra indicato).
2. Nel menu del browser, fare clic su **File**.
3. Fare clic su **Salva con nome**.
4. Andare all'indirizzario nel quale si desidera salvare il PDF.
5. Fare clic su **Salva**.

## Scaricare le informazioni su IBM Toolbox per Java in un pacchetto compresso

E' possibile scaricare un pacchetto compresso relativo all'argomento IBM Toolbox per Java che includa i javadoc nel sito Web [IBM Toolbox per Java e JTOpen](#) .

**Nota:** le informazioni su **IBM Toolbox per Java** hanno collegamenti ai documenti non inclusi nel pacchetto compresso. Questi collegamenti non funzioneranno nei file che si scaricano nella stazione di lavoro.

# Informazioni preliminari 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:

## [»Gestire l'installazione](#)

Vedere come modalità differenti di installazione e gestione dell'installazione di Toolbox per Java hanno effetto sulla facilità di gestione e sulle prestazioni.◀

## [Installare Toolbox per Java](#)

Acquisire informazioni sui requisiti per la funzionalità di OS/400 e della stazione di lavoro, prima di installare Toolbox per Java in un ambiente client/server. Acquisire conoscenze sulla modalità di installazione di Toolbox per Java sui server iSeries e sulle 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.

## [»Semplici esempi di programmazione](#)

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

# Gestire l'installazione di IBM Toolbox per Java

E' necessario installare IBM Toolbox per Java solo su sistemi client che lo utilizzano o in un'ubicazione sulla rete a cui i client possano 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 Toolbox per Java sulla partizione del client del server.

E' possibile utilizzare uno qualsiasi dei seguenti metodi (soli o in combinazione) per installare e gestire Toolbox per Java:

- [Gestione individuale](#) per installare e gestire individualmente Toolbox per Java su ogni client
- [Gestione della rete delle installazioni client](#) utilizzando AS400ToolboxInstaller per installare e gestire Toolbox per Java su ogni client
- [Gestione di rete di un'installazione singola](#) utilizzando la rete per installare e gestire un'installazione singola e condivisa di 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 installazioni Toolbox per Java su singoli client. Il vantaggio principale dell'installazione di Toolbox per Java su singoli client è la riduzione del tempo che il client impiega per avviare un'applicazione che utilizza le classi Toolbox per Java.

Lo svantaggio principale è costituito dalla gestione individuale di tali installazioni. Un utente o un'applicazione creata dall'utente devono tenere traccia e gestire la versione di Toolbox per Java installata su ogni stazione di lavoro.

## Gestione della rete delle installazioni del client

E' possibile scegliere di utilizzare la rete e [AS400ToolboxInstaller](#) per gestire le installazioni client di Toolbox per Java. Dal momento che ogni client dispone della propria copia di Toolbox per Java, questo tipo di installazione presenta lo stesso vantaggio di ridurre il tempo che un client impiega per avviare un'applicazione Toolbox per Java. Inoltre fa in modo di aggiornare automaticamente tutte le installazioni individuali di 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 un'installazione singola

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

- Tutti i client utilizzano la stessa versione di IBM Toolbox per Java
- L'aggiornamento dell'installazione singola di Toolbox per Java reca vantaggio a 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 Toolbox per Java. E' necessario inoltre abilitare il CLASSPATH del client in modo che punti al server. E' possibile utilizzare [iSeries NetServer](#), integrato ad OS/400 o un metodo differente che abiliti l'accesso ai file su iSeries NetServer, come ad esempio [iSeries Access per Windows](#).

# Installare 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](#)

## Installare Toolbox per Java

Dopo aver deciso come gestire l'installazione e dopo aver considerato i requisiti per l'esecuzione di IBM Toolbox per Java, è possibile installare Toolbox per Java:

- [Installare Toolbox per Java sul server](#)
- [Installare 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 delle applicazioni ToolboxME per iSeries«](#)

**Nota:** prima di utilizzare Toolbox per Java, assicurarsi di considerare i [requisiti della stazione di lavoro](#) che si riferiscono all'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](#).

## Avviare le opzioni OS/400 necessarie

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 **STRTCPVR SERVER(\*DDM)** e premere **INVIO**.

# 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 Toolbox per Java è già installato, completare le seguenti fasi:

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

**Nota:** è inoltre possibile scoprire se Toolbox per Java è installato utilizzando il comando CL (**GO MENU(LICPGM)**), Opzione 11.

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

Se è installata una versione precedente di Toolbox per Java, prima cancellare la versione installata attualmente poi [installare il prodotto su licenza IBM Toolbox per Java](#). Per evitare possibili problemi, sarebbe necessario fare una copia di riserva della versione di Toolbox per Java attualmente installata prima di cancellarla.

# Verificare il 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.

## Verificare il 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](#).



# Modificare il 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)`.
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 ci si collega a V4R4 o a 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 (Gestore certificati digitali)
- IBM Cryptographic Access Provider 128-bit per iSeries, 5722-AC3
- iSeries Client Encryption (128-bit), 5722-CE3

La versione V5R2 di IBM Toolbox per Java richiede l'utilizzo  della versione V5R1 o V5R2 dell'iSeries Client Encryption. 

Per ulteriori informazioni su SSL, consultare [Secure Sockets Layer e Java Secure Socket Extension](#).

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

Se si desidera utilizzare applet, servlet, SSL o la classe AS400ToolboxInstaller sul sistema iSeries, è necessario installare l'HTTP Server e installare i file classe sul sistema iSeries. Per ulteriori informazioni su IBM HTTP Server, consultare IBM HTTP Server for AS/400 Webmaster's Guide, GC41-5434, al seguente indirizzo URL:

<http://www.ibm.com/eserver/series/products/http/docs/doc.htm> . La guida Webmaster è disponibile sia in formato HTML che PDF.

Per informazioni sul Gestore certificati digitali e sulla modalità di creazione e gestione di certificati digitali utilizzando l'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.

## Eseguire IBM Toolbox per Java, Versione 5 Rilascio 2, sui server

Per installare IBM Toolbox per Java (programma su licenza 5722-JC1 V5R2M0) su un sistema iSeries, sul sistema deve essere in esecuzione almeno uno dei seguenti:

- OS/400 Versione 5 Rilascio 2
- OS/400 Versione 5 Rilascio 1
- OS/400 Versione 4 Rilascio 5

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.

## Utilizzare 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 la Versione 5 Rilascio 2 di IBM Toolbox per Java per accedere ai dati e alle risorse su un sistema iSeries, sul **server a cui ci si sta collegando** deve essere in esecuzione almeno uno dei seguenti:

- OS/400 Versione 5 Rilascio 2
- OS/400 Versione 5 Rilascio 1
- OS/400 Versione 4 Rilascio 5

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.

Modifica di Toolbox	Viene fornito con OS/400	LPP	Si installa su OS/400	Ritorna a collegarsi ad OS/400
Mod 0	V4R2	5763-JC1 V3R2M0	V3R2 e precedente	V3R2 e precedente
Mod 1	V4R3	5763-JC1 V3R2M1	V3R2 e precedente	V3R2 e precedente
Mod 2	V4R4	5769-JC1 V4R2M0	V4R2 e precedente	V4R2 e precedente
Mod 3	V4R5	5769-JC1 V4R5M0	V4R3 e precedente	V4R2 e precedente
Mod 4	V5R1	5722-JC1 V5R1M0	V4R4 e precedente	V4R3 e precedente
➤Mod 5	V5R2	5722-JC1 V5R2M0	V4R5 e precedente	V4R5 e precedente◀

# 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 una migliore prestazione, impostare le [proprietà dell'unità di controllo JDBC](#) per utilizzare una unità di controllo nativa quando il programma Java e il file database si trovano sullo stesso sistema 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 tabella che segue mostra quale versione di IBM Toolbox per Java e OS/400 è necessario 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	Modica Toolbox richiesta per utilizzare le ottimizzazioni native				
V4R2	Nessun potenziamento Mod0 disponibile.				
V4R3	Mod1	Mod2			
V4R4	Mod1	Mod2			
V4R5			Mod3		
V5R1				Mod4	
»V5R2					Mod5 «

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 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:

- [MEServer](#), incluso in IBM Toolbox per Java o nell'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 Toolbox per Java, assicurarsi di considerare i [requisiti OS/400](#) che si riferiscono al proprio ambiente.

# Requisiti della stazione di lavoro per eseguire le applicazioni Toolbox per Java

Per sviluppare ed eseguire le applicazioni Toolbox per Java, assicurarsi che la stazione di lavoro soddisfi i requisiti che seguono:

- Si consiglia di utilizzare una JVM (Java virtual machine) che supporti Java 2 Platform, Standard Edition (J2SE<sup>(TM)</sup>) o Enterprise Edition (J2EE<sup>(TM)</sup>), versioni 1.3.x o superiori. Molte nuove funzioni Toolbox per Java richiedono l'utilizzo di questa versione della JVM. Tuttavia, si può comunque utilizzare una JVM che supporta completamente JDK 1.1.8 o un successivo JDK, incluso Java 2 Platform.
- Se il programma utilizza il Graphical Toolbox o le classi nel pacchetto vaccess, è richiesto anche [Swing 1.1](#). Gli ambienti che seguono sono stati esaminati:
  - »Windows<sup>(R)</sup> 2000«
  - »Windows<sup>(R)</sup> XP«
  - AIX Versione 4.3.3.1
  - Sun Solaris<sup>(TM)</sup> Versione 5.7
  - »OS/400 Versione 4 Rilascio 5 o successive«
  - »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 le applicazioni Toolbox per Java, assicurarsi che la stazione di lavoro soddisfi i requisiti che seguono:

- 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 eseguire l'applet che utilizza le classi Toolbox per Java in un browser, è necessario installare un modulo aggiuntivo come ad esempio il [modulo aggiuntivo Sun Java 2 Runtime](#)


[Environment \(JRE\) 1.3.0](#)  «

- TCP/IP installato e configurato
- » La stazione di lavoro si deve collegare ad un server su cui è in esecuzione OS/400 V4R5 o successive «



# 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 rilascio 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 rilasci 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.](http://www.sun.com) 

# Installare 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 Toolbox per Java viene fornita con OS/400. In questo modo, se si desidera utilizzare Toolbox per Java solo sul server iSeries, non è necessario installare il prodotto su licenza. Per ulteriori informazioni sulla versione origine di Toolbox per Java, consultare [File jar: nota 1](#).

Prima di installare IBM Toolbox per Java, è necessario assicurarsi che la versione di OS/400 soddisfi i [requisiti per l'esecuzione di Toolbox per Java](#). Inoltre, alcuni server vengono forniti preconfigurati con un'installazione di Toolbox per Java. Si può avere l'esigenza di [determinare se il prodotto su licenza Toolbox per Java sia già installato](#) sul server.

## Installare Toolbox per Java

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

### Utilizzare iSeries Navigator per installare Toolbox per Java

Per installare Toolbox per Java utilizzando iSeries Navigator, completare le fasi che seguono:

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 in cui si desidera installare 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 casella 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.

### Utilizzare la riga comandi per installare Toolbox per Java

Per installare Toolbox per Java da una riga comandi iSeries, completare le fasi che seguono:


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 [Gestire software e programmi su licenza](#).

# Installare 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 Toolbox per Java su client singoli, copiare i file JAR sulla stazione di lavoro e configurare la stazione di lavoro CLASSPATH.
- Per utilizzare il Toolbox per Java installato sul server, è necessario semplicemente configurare il CLASSPATH della stazione di lavoro in modo da puntare all'installazione del 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 Toolbox per Java nell'applicazione richiede inoltre che il sistema soddisfi i [requisiti per OS/400](#).

I file classe Toolbox per Java sono contenuti in diversi file jar, di conseguenza l'utente ha bisogno di copiare uno o più di questi file jar sulla stazione di lavoro. Per ulteriori informazioni relative a quali file jar sono richiesti per funzioni Toolbox per Java specifiche, consultare [File jar](#).


## Esempio: copiare 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 ciò in diversi 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 per la 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 di utilizzare la versione origine aperta di Toolbox per Java, denominata JTOpen. Per ulteriori informazioni relative a JTOpen, consultare [il sito web IBM Toolbox per Java and JTOpen](#) .


Se il browser non è abilitato a Javascript, fare clic sul collegamento al testo accanto a Immagine da collegare alla nota appropriata; le note sono tutte elencate nella tabella seguente.



























## 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 tabella che segue indica quali file jar devono trovarsi in CLASSPATH per utilizzare classi dal pacchetto elencato.



Alcune immissioni nella tabella sono dotate di note per offrire ulteriori informazioni. Se il browser è abilitato a Javascript, fare clic sull'immagine  per visualizzare le informazioni su un'altra finestra. Altrimenti, è possibile fare clic sul collegamento testo per collegarsi alle stesse informazioni elencate nella tabella seguente.

Pacchetto o funzioni Toolbox per Java	File jar necessari in CLASSPATH
<a href="#">Classi Access</a>	jt400.jar (client) o jt400Native.jar (server) nel solito ambiente client/server  <a href="#">Nota 1</a> , o jt400Proxy.jar in un ambiente proxy
<a href="#">CommandPrompter</a>  <a href="#">Nota 2</a>	jt400.jar, jui400.jar, util400.jar  <a href="#">Nota 3</a> e x4j400.jar 
<a href="#">Classi HTML</a>	jt400.jar  <a href="#">Nota 1</a> più jt400Servlet.jar (client) o jt400Native.jar (server)  <a href="#">Nota 1</a>
GUI sorgente dati JDBC  <a href="#">Nota 4</a>	jt400.jar (client)  <a href="#">Nota 1</a> e jui400.jar
Sistema NLS e messaggi di errore	jt400Mri_lang_cntry.jar  <a href="#">Nota 5</a>
PCML (sviluppo)  <a href="#">Nota 6</a>	jt400.jar (client) o jt400Native.jar (server)  <a href="#">Nota 1</a> ,  <a href="#">Nota 7</a> , e x4j400.jar
PCML (tempo di esecuzione, serializzato)	jt400.jar (client) o jt400Native.jar (server)  <a href="#">Nota 1</a> ,  <a href="#">Nota 7</a>
PDML (sviluppo)  <a href="#">Nota 2</a>	uitools.jar, jui400.jar, util400.jar  <a href="#">Nota 3</a> , e x4j400.jar
PDML (tempo di esecuzione, analizzato)  <a href="#">Nota 2</a>	jui400.jar, util400.jar  <a href="#">Nota 3</a> , e x4j400.jar
PDML (tempo di esecuzione, serializzato)  <a href="#">Nota 2</a>	jui400.jar e util400.jar  <a href="#">Nota 3</a>
<a href="#">Classi ReportWriter</a>	jt400.jar (client) o jt400Native.jar (server)  <a href="#">Nota 1</a> e reportwriter jars  <a href="#">Nota 8</a>
<a href="#">Classi Resource</a>	jt400.jar (client) o jt400Native.jar (server)  <a href="#">Nota 1</a>
<a href="#">RFML</a>	jt400.jar (client) o jt400Native.jar (server)  <a href="#">Nota 1</a> , e x4j400.jar 
<a href="#">Classi di sicurezza</a>	jt400.jar (client) o jt400Native.jar (server) nel solito ambiente client/server  <a href="#">Nota 1</a> , o jt400Proxy.jar in un ambiente proxy

<a href="#">Classi Servlet</a>	jt400.jar  <a href="#">Nota 1</a> più jt400Servlet.jar (client) o jt400Native.jar (server)  <a href="#">Nota 1</a>
<a href="#">Programma di debug sistema</a> <a href="#">iSeries</a>  <a href="#">Nota 2</a>	jt400.jar (client)  <a href="#">Nota 1</a> e tes.jar 
<a href="#">ToolboxME per iSeries</a>	jt400Micro.jar (client) e jt400.jar (server) o jt400Native.jar (server)  <a href="#">Nota 9</a> 
<a href="#">Classi Vaccess</a>	jt400.jar (client)  <a href="#">Nota 1</a>

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

- **jt400.jar** - supporto access, resource, vaccess, security, PCML, RFML, JDBC e MEServer. 
- **jt400.zip** - utilizzare jt400.jar invece di jt400.zip. jt400.zip viene fornito per mantenere la compatibilità con i rilasci precedenti di IBM Toolbox per Java.
- **jt400Access.zip** - access, resource, security, PCML; le stesse classi che si trovano in jt400.jar ad eccezione delle classi vaccess. jtAccess400.zip viene fornito per mantenere la compatibilità con i rilasci precedenti di IBM Toolbox per Java. Utilizzare jt400.jar o jt400Native.jar invece di jt400Access.zip.
- **jt400Native.jar** - access, resource, security, PCML, HTML, RFML, MEServer 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:** l'utilizzo di CommandPrompter, PDML o del programma di debug del sistema iSeries richiede un file jar aggiuntivo che non sia parte di Toolbox per Java: jhall.jar. Per ulteriori informazioni relative allo scaricamento di jhall.jar, consultare il sito web [Sun JavaHelp\(TM\)](#)  

**Nota 3:** util400.jar contiene classi iSeries specifiche per formattare l'immissione e per utilizzare il programma di richiesta della riga comandi (CL). L'utilizzo della classe CommandPrompter richiede util400.jar. Invece, per utilizzare PDML non è necessario util400.jar, comunque lo si consiglia.

**Nota 4:** 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 5:** jt400Mri\_xx\_yy.jar contiene messaggi convertiti, incluse stringhe contenute in messaggi di eccezione, finestre di dialogo e immissioni 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'installazione 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 6:** 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 [Creazione di chiamate del programma iSeries con PCML](#).

**Nota 7:** 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 rilasci precedenti di IBM Toolbox per Java.

**Nota 8:** le copie delle classi ReportWriter si trovano in più di un file jar:

- composer.jar
- outputwriter.jar
- reportwriters.jar
- xslparser.jar
- x4j400.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 9:** 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 dal 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 `Property`**

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", "hgoffice");
    }
}

```

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 del server proxy

Proprietà del server proxy	Descrizione
<code>com.ibm.as400.access.AS400.proxyServer</code>	Specifica il nome host del server proxy e il numero porta, utilizzando il formato:  <code>hostName:portNumber</code> Il numero porta è facoltativo.
<code>com.ibm.as400.access.SecureAS400.proxyEncryptionMode</code>	Specifica quale parte del flusso di dati proxy viene codificata utilizzando l'SSL. I valori validi sono: <ul style="list-style-type: none"> <li>● 1 = Client proxy sul server proxy</li> <li>● 2 = Server proxy su iSeries</li> <li>● 3 = Client proxy sul server proxy e il server proxy su iSeries</li> </ul>
<code>com.ibm.as400.access.TunnelProxyServer.clientCleanupInterval</code>	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.
<code>com.ibm.as400.access.TunnelProxyServer.clientLifetime</code>	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 rilevare 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 di traccia

Proprietà di traccia	Descrizione
com.ibm.as400.access.Trace.category	Specifica le categorie di traccia da abilitare. Questa è una lista delimitata da virgola che contiene tutte le combinazioni delle categorie di traccia. La lista completa delle categorie di traccia viene definita nella classe <a href="#">Trace</a> .
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 informazioni sui valori supportati, consultare la <a href="#">Proprietà della traccia del server JDBC</a> .«


## Proprietà CommandCall/ProgramCall

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

# Esempi semplici di programmazione

Questi esempi mostrano alcuni modi per codificare i propri programmi Java utilizzando le classi IBM Toolbox per Java. Sono rivolti ai programmatori con poca esperienza con le classi Toolbox per Java, questi esempi includono informazioni dettagliate sulle righe chiave nel codice.

E' possibile visualizzare le informazioni dettagliate nei seguenti modi:

- Fare clic sull'immagine  per visualizzare le informazioni dettagliate in una finestra a comparsa. (Per visualizzare le note in una finestra a comparsa, il browser deve supportare JavaScript.)
- Fare clic sul collegamento testo per visualizzare le informazioni dettagliate fornite alla fine dell'esempio.

Se si desidera avere supporto nella fase iniziale, consultare [Come scrivere il primo programma Toolbox per Java](#).

Per i collegamenti ad altri esempi forniti con le informazioni relative a Toolbox per Java, consultare [Esempi di codice](#).

Utilizzare la lista seguente per visualizzare gli esempi semplici di programmazione:

- [Chiamare i comandi](#)
- [Utilizzare le code messaggi](#)
- [Utilizzare l'accesso a livello record](#)
- [Utilizzare le classi JDBC per creare e riempire una tabella](#)
- [Visualizzare una lista 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**

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 implicare 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.

# 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 è chiamato semplicemente pacchetto Access.

Utilizzare i collegamenti nella seguente lista per trovare informazioni sulle classi in differenti pacchetti Toolbox per Java:

- [Classi Access](#) abilitano all'accesso e alla gestione delle risorse su iSeries
- [Classi HTML](#) consentono di creare velocemente moduli e tabelle HTML
- [Classi Micro](#) abilitano alla creazione di 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 di sicurezza](#) proteggono i collegamenti con il server e verificano l'identità di un utente che lavora sul sistema iSeries
- [Classi Servlet](#) assistono nel richiamo e nella formattazione dei dati per l'utilizzo dei servlet Java
- [Classi programma di utilità](#) 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 di IBM Toolbox per Java

Le classi Access (di accesso) IBM Toolbox per Java rappresentano dati e risorse iSeries. Le [classi gestiscono i server iSeries e AS/400e](#) per fornire un'interfaccia abilitata ad internet per accedere ed aggiornare dati e risorse del server.

Le seguenti classi forniscono l'accesso alle risorse di iSeries e AS400e:

- [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 sistema 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 al lotto di sistema

informazioni

- [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:** Toolbox per Java fornisce una seconda serie di classi, chiamate [classi risorsa](#), per gestire gli oggetti e le liste iSeries. Le classi delle risorse presentano una framework generica e un'interfaccia di programmazione coerente per gestire vari oggetti e liste iSeries. Dopo aver letto informazioni relative alle classi nel [pacchetto Access](#) e nel [pacchetto resource](#), è possibile scegliere l'oggetto più adatto all'applicazione.

# 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 a iSeries. Ad esempio, l'oggetto CommandCall richiede un oggetto AS400 prima che possa inviare i comandi a 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](#).

»A questo punto gli oggetti AS400 supportano l'autenticazione Kerberos, utilizzando l'API (Application Programming Interface) JGSS (Java Generic Security Service) per autenticare il 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 [J2SDK, v1.4 Security Documentation](#)  «

Consultare [Gestire i collegamenti](#) per le informazioni sulla gestione dei collegamenti a 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:

- [Autenticare](#) il profilo dell'utente
- » [Richiamare la credenziale di un token di profilo](#) e autenticare il profilo utente associato «
- » [Impostare la credenziale di un token di profilo](#) «
- [Gestire gli ID utenti predefiniti](#)
- [Memorizzare le parole d'ordine nella cache](#)
- [Richiedere l'ID utente](#)
- [Modificare](#) una parola d'ordine
- Richiamare [la versione](#) e il [rilascio](#) dell'iSeries

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

# 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 un 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

L'esempio che segue mostra come utilizzare CommandCall per inviare comandi al sistema iSeries utilizzando un collegamento sicuro:

```
// Creare un oggetto AS400 sicuro. 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 sicuro viene 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:

- [Eseguire il ping del server](#)
- [Eseguire il ping di uno specifico servizio](#) sul server
- [Impostare un oggetto PrintWriter](#) a cui si desidera collegare le informazioni sul ping
- [Impostare la verifica del supero tempo](#) per un'operazione ping

**Esempio:** utilizzando 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");
```



# 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: utilizzare la classe AS400BidiTransform per convertire un testo bidirezionale

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");
```



# 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 comprendano 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 [Configurare il 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 una lista di voci](#) da una tabella ad accesso casuale suddivisa in cluster per tutti i profili utente

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 [Gestire i collegamenti](#).

# Command call

La classe [CommandCall](#) consente ad un programma Java di richiamare un comando iSeries non interattivo. I risultati del comando sono disponibili in una lista 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.

Il seguente esempio indica come utilizzare la classe CommandCall per eseguire un comando su un sistema 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
// successivamente. Potrebbe impostarlo 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);
```

Utilizzando la classe CommandCall l'oggetto AS400 si collega a iSeries. Consultare [Gestire i collegamenti](#) per informazioni sulla gestione dei collegamenti.

» Quando il programma Java e il comando del server iSeries sono sullo stesso server, il comportamento predefinito di Toolbox per Java è quello di cercare la sicurezza del sottoprocesso per il comando nel sistema. « Se protetto dal sottoprocesso, il comando è 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\(\)](#) .

## Esempio

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 è disponibile per essere utilizzato da un programma Java come parte di supporto Toolbox per il JDBC 2.0 Optional Package API. ➤ L'interfaccia JDBC ConnectionPool è anche supportata in JDBC 3.0 API, il quale è legato a Java 2 Platform, Edizione Standard, 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 accrescere le prestazioni evitando tentativi di collegamento ripetuti. 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 dell'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 [Gestire i collegamenti](#) per ulteriori informazioni sulla gestione quando viene aperto un collegamento con iSeries durante l'utilizzo delle classi AS400ConnectionPool.

**Esempio:** Utilizzare [AS400ConnectionPool](#) per riutilizzare 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 gli [attributi](#) dell'area dati
- Impostare il [sistema](#) dove si trova l'area dati

L'utilizzo della classe DataArea collega l'oggetto AS400 al server. Consultare [Gestire i 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 del percorso dell'IFS](#) dell'oggetto rappresentato dall'area dati.
- [Leggere](#) tutti i dati che sono contenuti nell'area dati
- Leggere una [quantità 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 essa si trova
- 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

Il seguente esempio mostra come creare e scrivere in un'area dati decimali:

```
// 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 classiDataArea. 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 classiDataArea. 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.

## 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 \(numerico e testo\)](#)

## 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

Due esempi illustrano l'utilizzo delle classi di conversione formato record con code dati:

- Esempio: [utilizzare le classi Record e RecordFormat per inserire dati in una coda](#)
- Esempio: [utilizzare le classi FieldDescription, RecordFormat e Record](#)

# Classi di conversione per dati numerici

Le classi di conversione per i dati numerici convertono semplicemente dati numerici dal formato utilizzato sul server iSeries o AS/400e (chiamato **formato server** nella seguente tabella) nel formato Java. La seguente tabella mostra i tipi supportati:

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

Il seguente esempio mostra la conversione da un tipo numerico nel formato server ad un int Java:

```
// Creare un buffer per conservare il tipo dati
// server. Supponiamo che il buffer sia occupato da
// dati numerici nel formato server da
// code dati, chiamata al programma e così via.
byte[] data = new byte[100];

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

// Convertire dal tipo di server all'oggetto
// Il numero parte dall'inizio del
// del buffer.
Integer intObject = (Integer) bin4Converter.toObject(data,0);

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

Il seguente esempio mostra la conversione da un int Java ad un tipo dati numerici nel 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 del server.
```

```
AS400Bin4 bin4Converter = new AS400Bin4();

        // Convertire da oggetto Java al tipo dati
        // del server.
byte[] data = bin4Converter.toBytes(intObject);

        // Rilevare quanti byte del buffer
        // erano stati occupati dal valore
        // del 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 »13488 Unicode.« Il metodo [toBytes\(\)](#) converte dal formato Java alla schiera di byte nel 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.

Ad esempio, supponiamo che un oggetto DataQueueEntry restituisca il testo iSeries in EBCDIC. Il seguente esempio converte questi dati in Unicode in modo che il programma Java possa utilizzarli:

»

```
// ... Supponiamo che il lavoro coda dati sia già stato svolto per
// reperire 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. Si
noti che si sta creando un programma
// di conversione predefinito. Questo programma di conversione
presuppone 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 il programma
sia in esecuzione
// come un Toolbox per un client proxy Java.
int ccsid = 37;
AS400 system = ...; //oggetto AS400
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 in 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.

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 tipi di dati
        // che corrisponda ad una struttura
        // contenente:
        //     - 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 una conversione oggetto 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 (lasciare
caratt. di 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. Anche i dati restituiti saranno una
        // una schiera di byte.

        // Convertire i dati restituiti dal formato
        // iSeries in quello Java.
Object[] myRoundTripData =
    (Object[])myConverter.toObject(myAS400Data,0);

        // Estrarre 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 i dati provenienti dall'accesso a livello record, esso può anche specificare ogni parola chiave DDS (data definition specification) di iSeries o AS/400e che descrive ulteriormente il campo.

Le classi FieldDescription sono le seguenti:

- [BinaryFieldDescription](#)
- [CharacterFieldDescription](#)
- [DateFieldDescription](#)
- [DBCSEitherFieldDescription](#)
- [DBCSTextFieldDescription](#)
- [DBCSTGraphicFieldDescription](#)
- [DBCSTOnlyFieldDescription](#)
- [DBCSTOpenFieldDescription](#)
- [FloatFieldDescription](#)
- [HexFieldDescription](#)
- [PackedDecimalFieldDescription](#)
- [TimeFieldDescription](#)
- [TimestampFieldDescription](#)
- [ZonedDecimalFieldDescription](#)

Ad esempio, supponiamo 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). Questi possono essere descritti dalle seguenti descrizioni campo:

```
// Creare una descrizione campo relativa
// ai dati numerici. Si noti che utilizza
// il tipo di dati AS400Bin4. Ridenomina inoltre
// il campo affinché vi si possa accedere
// a seconda del nome nella classe
// record.
BinaryFieldDescription bfd = new BinaryFieldDescription(new
AS400Bin4(),
                                                                    "msgNumber");

// Creare una descrizione campo relativa
// ai dati carattere. Nota: utilizza
// il tipo dati AS400Text. Ridenomina inoltre
// il campo affinché vi si possa accedere
// a seconda del nome dalla classe
// record.
CharacterFieldDescription cfd1 = new CharacterFieldDescription(new
AS400Text(8),
"msgTime");

// Creare una descrizione campo relativa
// ai dati carattere. Nota: utilizza
// il tipo dati AS400Text. Ridenomina inoltre
```



```
        // il campo affinché vi si possa accedere
        // a seconda del nome dalla classe
        // record.
        CharacterFieldDescription cfd2 = new CharacterFieldDescription(new
AS400Text(50),

"msgText" );
```

Le classi FieldDescription sono pronte ora per essere raggruppate in una classe RecordFormat. L'esempio continua nella sezione [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.

Ad esempio, per aggiungere le descrizioni di campo create nell'esempio [descrizione campo](#) ad un formato record:

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

Il programma è ora pronto per creare un record da un formato record. L'esempio continua nella sezione [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](#).

Ad esempio, per utilizzare il formato record creato nell'esempio [Formato record](#):

```

// Supponiamo che il lavoro di impostazione della
coda dati sia
// già stato svolto. Ora leggere un
// record dalla coda dati.
DataQueueEntry dqe = dq.read();

// I dati dalla coda dati si trovano
// ora in una voce di coda dati. Estrarre
// i dati dalla voce coda dati
// ed inserirli nel record.
// Si ottiene un record predefinito
// dall'oggetto formato record ed
// inizializzarlo con i dati dalla
// voce coda dati.
Record dqRecord = rf.getNewRecord(dqe.getData());

// Ora che i dati si trovano nel
// record, estrarre i dati da un
// campo per volta, convertendo i
// dati mentre vengono rimossi. Il risultato
// è costituito da dati in un oggetto Java che
// il programma ora può elaborare.
Integer msgNumber = (Integer) dqRecord.getField("msgNumber");
String msgTime = (String) dqRecord.getField("msgTime");
String msgText = (String) dqRecord.getField("msgText");
```

# 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: utilizzare la classe LineDataRecordWriter

```
// 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.");
}
```

# Data queue

Le classi DataQueue consentono al programma Java di interagire con le code dati del server. Le code dati sui server iSeries e AS/400e posseggono le 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 [gestire i 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 [Nomi dei percorsi dell'IFS](#).

## 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 oggetto [KeyedDataQueueEntry](#) 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.

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

```
        // 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

Esempi di coda di dati sequenziali, nei quali il produttore immette elementi nella coda dati e il consumatore li estrae dalla coda e li elabora:

- Esempio produttore coda dati sequenziali .
- Esempio consumatore coda dati sequenziali .



## 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

Nei seguenti esempi di code dati con chiave, il produttore inserisce gli elementi in una coda dati e il consumatore estrae gli elementi dalla coda e li elabora:

- Esempio produttore coda dati con chiave
- Esempio consumatore coda dati con chiave

# Certificati digitali


I Certificati digitali sono istruzioni firmate digitalmente 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 and sicurezza di Java, consultare all'indirizzo [la pagina Sun Microsystems, Inc., Java Security](#) ) Il programma su licenza IBM Toolbox per Java dispone di classi che gestiscono i certificati digitali sul server iSeries o AS/400e.

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 la lista di convalida o il profilo utente.
- Gestire certificati, ad esempio, aggiungere un certificato al profilo utente o cancellare un certificato da una lista di convalida.

Utilizzando una classe di certificato si collega l'oggetto AS400 al server. Consultare [Gestire i collegamenti](#) per informazioni sulla gestione dei collegamenti.

Sul server, i certificati appartengono ad una lista di convalida o a un profilo utente.

- La classe [AS400CertificateUserProfileUtil](#) dispone dei metodi per la gestione dei certificati su un profilo utente.
- La classe [AS400CertificateVldUtil](#) dispone dei metodi per la gestione dei certificati in una lista di convalida.

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.

## Lista dei certificati

Per richiamare una lista 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 la lista dei certificati in una lista 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 alla lista. Ad esempio, una lista 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 lista 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.

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

```
        // 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 attributo
        // certificato. Si richiedono solo certificati
        // per una singola persona in modo che l'elenco
        // sia composta da un solo elemento.
AS400CertificateAttribute[] attributeList = new
AS400CertificateAttribute[1];
    attributeList[0] = new
AS400CertificateAttribute(AS400CertificateAttribute.SUBJECT_COMMON_NAME,
"Jane Doe");

        // Richiamare l'elenco che soddisfa
        // i criteri. Lo spazio utente "myspace"
        // nella libreria "mylib" verrà utilizzato
        // per la memorizzazione dei certificati.
        // E' necessario che vi sia spazio utente
        // prima di chiamare 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 d'ambiente sono diverse dai valori del 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: creare, impostare e richiamare variabili d'ambiente

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

```
// 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 tre informazioni:

- **Tipo di errore** - L'oggetto eccezione lanciato 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 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.

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

```
        // ... tutto il lavoro di impostazione per cancellare
        // un file sul server tramite la
        // classe IFSFile è stato svolto. Ora tentare
        // la cancellazione del file.
            try
    {
        aFile.delete();
    }

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

        // Richiamare il codice di ritorno
        // dall'eccezione e visualizzare ulteriori
        // informazioni basate sul 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 tenere 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 a parte. 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

Ad esempio, con la classe FTP, è possibile [copiare](#) un insieme di file da un indirizzario su un 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 FTP

Mentre la classe FTP è un'interfaccia FTP generica, la [sottoclasse AS400FTP](#) viene scritta specificamente per il server FTP sul server. Riconosce le semantiche del server FTP sul server iSeries o AS/400e, in modo che non lo debba fare il programmatore. 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.

L'esempio che segue immette un file di salvataggio nel server. Tenere presente che l'applicazione non imposta il tipo di trasferimento dati su binario né utilizza Toolbox 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 o AS/400e 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.

In aggiunta a questi metodi, le classi contengono metodi per eseguire queste operazioni:

- 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 [gestire i 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
```

Le classi IFS sono:

Classe IFS	Descrizione
<a href="#">IFSFile</a>	Rappresenta un file in IFS
<a href="#">IFSJavaFile</a>	Rappresenta un file in IFS (estende java.io.File)
<a href="#">IFSFileInputStream</a>	Rappresenta un flusso di immissioni per la lettura di dati da un file iSeries
<a href="#">IFSTextFileInputStream</a>	Rappresenta un flusso di dati carattere letti da un file
<a href="#">IFSFileOutputStream</a>	Rappresenta un flusso di emissioni per la scrittura di dati su un file iSeries
<a href="#">IFSTextFileOutputStream</a>	Rappresenta un flusso di dati carattere scritti su un file
<a href="#">IFSRandomAccessFile</a>	Rappresenta un file su iSeries per la lettura e la scrittura di dati
<a href="#">IFSFileDialog</a>	Consente di spostarsi nel file system e di selezionare un file in esso



## Esempi

L'[esempio IFSCopyFile](#) mostra come utilizzare le classi IFS per copiare un file da un indirizzario all'altro su iSeries.

L'[esempio File List](#) mostra come utilizzare le classi IFS per elencare il contenuto di un indirizzario su 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 la lista dei 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:** [creare un indirizzario](#)
- **Esempio:** [utilizzare eccezioni per la traccia degli errori](#)
- **Esempio:** [lista dei file con un'estensione .txt](#)
- **Esempio:** [utilizzare listFiles\(\) per elencare il contenuto di un indirizzario](#)

# 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 la lista dei 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 essere necessario aggiornare i dati nuovamente.

Di seguito è riportato un esempio di 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 e AS/400e. 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.

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();

// Allocare 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();
```

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](#).

# 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 sono
        // convertiti in Unicode da
        // IFSTextFileInputStream.
System.out.println(s);

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

# 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.

L'esempio che segue 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();
```

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](#).

# 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`.

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` o `AS/400e`. 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.

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 della dimensione della
    // scrittura.
    aFile.write(i);
}

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

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.

## IFSFileDialog

La classe [IFSFileDialog](#) consente di consultare il file system e di selezionare un file. Questa classe utilizza la classe [IFSFile](#) per consultare la lista degli indirizzi e dei file nell'IFS sul server iSeries o AS/400e. 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.

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 una lista 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");

// Mostrare 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).

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. Questo
// 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 può
// 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 operazioni sul
// file.

// Chiudere il file. Ora le 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

Questo [esempio](#) mostra come eseguire un programma sul server dal client che emette "Hello World!".

# JDBC

JDBC<sup>(TM)</sup> 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' possibile utilizzare anche l'[unità di controllo JDBC di IBM Developer Kit per Java](#), denominata unità di controllo JDBC 'nativo':

- Utilizzare l'unità di controllo JDBC 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 sistema iSeries

Per ulteriori informazioni sugli aggiornamenti in corso, consultare [Novità in V5R2](#) e [Potenziamenti del supporto JDBC di Toolbox per Java](#).⏪

## 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.
- ➤Le API 3.0 di JDBC (pacchetti java.sql and javax.sql) sono incluse in J2SE, Versione 1.4.⏪

## Interfacce supportate

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

Interfaccia JDBC supportata	API necessaria
<a href="#">Blob</a> fornisce l'accesso ai BLOB	JDBC 2.1 principale
<a href="#">CallableStatement</a> esegue le procedure memorizzate SQL	JDK 1.1
<a href="#">Clob</a> fornisce l'accesso ai CLOB	JDBC 2.1 principale
<a href="#">Connection</a> rappresenta un collegamento ad un determinato database	JDK 1.1
➤ <a href="#">ConnectionPool</a> rappresenta un lotto di oggetti di collegamento.	JDBC 2.0 Optional Package ⏪
<a href="#">ConnectionPoolDataSource</a> rappresenta una produzione per gli oggetti di lotto AS400JDBCPooledConnection	JDBC 2.0 Optional Package
<a href="#">DatabaseMetaData</a> fornisce informazioni sull'intero database.	JDK 1.1
<a href="#">DataSource</a> rappresenta una produzione per i collegamenti al database.	JDBC 2.0 Optional Package
<a href="#">Il programma di controllo</a> crea il collegamento e restituisce informazioni sulla versione del programma di controllo.	JDK 1.1
➤ <a href="#">ParameterMetaData</a> fornisce la capacità di richiamare informazioni sui tipi e le proprietà dei parametri in un oggetto PreparedStatement	API di JDBC 3.0⏪
<a href="#">PreparedStatement</a> esegue istruzioni compilate SQL	JDK 1.1

<a href="#">ResultSet</a> 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
<a href="#">ResultSetMetaData</a> fornisce informazioni su uno specifico ResultSet	JDK 1.1
<a href="#">RowSet</a> è una serie di righe connesse che contengono un ResultSet	JDBC 2.0 Optional Package
<a href="#">Savepoint</a> fornisce un controllo particolareggiato nelle transazioni	API di JDBC 3.0
<a href="#">Istruzione</a> esegue istruzioni SQL ed ottiene i risultati	JDK 1.1
<a href="#">XAConnection</a> è un collegamento database che partecipa alle transazioni XA globali	JDBC 2.0 Optional Package
<a href="#">XAResource</a> è un gestore risorse che si utilizza nelle transazioni XA	JDBC 2.0 Optional Package

E' stata inclusa una tabella che elenca le proprietà di JDBC per un facile riferimento.

## Esempi

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

- Utilizzo dell'unità di controllo JDBC per [creare e popolare](#) una tabella
- Utilizzo dell'unità di controllo JDBC per [interrogare](#) una tabella e generare un'emissione del suo contenuto





# Potenziamenti del supporto JDBC di Toolbox per Java

Le funzioni potenziate di JDBC per OS/400 Versione 5 Rilascio 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 di 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, 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 Toolbox per Java ora sono le stesse di quelle per il [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()` in [AS400JDBCStatement](#) 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 aggiornato 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 Toolbox per Java devono far corrispondere tutti i nomi di colonna forniti dall'utente o dall'applicazione con i nomi che si trovano nella tabella del database. In ogni caso, quando un nome di colonna non è contenuto tra gli apici, 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 con il nome sul server oppure Toolbox per Java invia un'eccezione.

## Specifiche 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](#) di Toolbox per Java (quando i metodi in 1. o

2. non sono utilizzati)

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 Toolbox per Java inviava un'eccezione al momento della commutazione a \*NONE dopo aver effettuato 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:** la lista seguente 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](#)
- [Proprietà del server](#)
- [Proprietà del formato](#)
- [Proprietà delle prestazioni](#)
- [Proprietà della funzione di ordinamento](#)
- [Altre proprietà](#)

## 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	Necessario	Opzioni	Valore predefinito
"password"	Specifica la parola d'ordine per il collegamento al server. Se non è specificata, sarà richiesta 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 deve essere chiesto se sono necessari un ID utente e una parola d'ordine per il collegamento al server. Se è impossibile effettuare 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 è specificata, sarà richiesta 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	Necessario	Opzioni	Valore predefinito
----------------------	-------------	------------	---------	--------------------

"congelamento cursore"	<p>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.</p>	no	"true" "false"	"true"
»"Sensibilità del cursore"	<p>Specifica la sensibilità del cursore richiesta dal database. La funzionalità dipende da resultSetType:</p> <ul style="list-style-type: none"> <li>● 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.</li> <li>● ResultSet.TYPE_SCROLL_INSENSITIVE fa in modo che questa proprietà venga ignorata.</li> </ul> <p>Questa proprietà viene ignorata quando ci si collega a sistemi su cui è in esecuzione la versione V5R1 e versioni precedenti di OS/400.</p>	no	"non sensibile" "insensibile" "sensibile"	"non sensibile"«
»"nome del database"	<p>Specifica il database da utilizzare per il collegamento, compreso quello memorizzato in un <a href="#">IASP (Independent Auxiliary Storage Pool)</a>. 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:</p> <ul style="list-style-type: none"> <li>● 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.</li> <li>● Quando questa proprietà viene utilizzata per specificare *SYSBAS come nome del database, si utilizza il database predefinito del sistema.</li> <li>● 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.</li> </ul>	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.«

"librerie"	<p>Specifica una o più librerie che si desidera aggiungere o sostituire nella lista librerie del lavoro server e facoltativamente imposta la libreria predefinita (schema predefinito).</p> <p><b>Lista librerie</b>  Il server utilizza le librerie specificate per definire i nomi procedura memorizzati non qualificati e le procedure memorizzate li utilizzano per definire i 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 la lista librerie corrente del lavoro server:</p> <p>Quando la prima voce è *LIBL, le librerie specificate vengono aggiunte alla lista librerie corrente. Quando non viene utilizzato *LIBL, le librerie specificato sostituiscono la lista corrente delle librerie del lavoro server</p> <p><b>Schema predefinito</b>  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.</p> <ul style="list-style-type: none"> <li>● Denominazione SQL  Quando non si specifica lo schema predefinito sull'URL del collegamento: <ul style="list-style-type: none"> <li>○ La prima voce (a meno che non sia *LIBL) diventa lo schema predefinito</li> <li>○ Quando la prima voce è *LIBL, la seconda voce diventa lo schema predefinito</li> <li>○ Quando non si imposta questa proprietà oppure quando contiene solo *LIBL, il profilo utente diventa lo schema predefinito</li> </ul> </li> <li>● Denominazione di sistema  Quando non si specifica lo schema predefinito sull'URL del collegamento: <ul style="list-style-type: none"> <li>○ Non viene impostato alcuno schema predefinito e il server utilizza le librerie specificate per la ricerca dei nomi non qualificati</li> <li>○ Quando questa proprietà non viene impostata o quando contiene solo *LIBL, il server utilizza la lista librerie corrente del lavoro server per la ricerca dei nomi non qualificati</li> </ul> </li> </ul>	no	Lista delle librerie server, separate da virgole o spazi	"*LIBL"
------------	--	----	--	---------

"isolamento transazione"	Specifica l'isolamento transazione predefinito.	no	"nessuna" "lettura non sincronizzata" "lettura sincronizzata" "lettura ripetibile" "serializzazione"	"lettura non sincronizzata"
--------------------------	---	----	--	-----------------------------

## 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	Necessario	Opzioni	Valore predefinito
"formato data"	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)
"separatore data"	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 "formato data" non sia impostata su "julian", "mdy", "dmy" o "ymd".	no	"/" (barra trasversale) "." (trattino) "." (punto) "," (virgola) "b" (spazio)	(lavoro del server)
"separatore decimale"	Specifica il separatore decimale utilizzato in costanti letterali numeriche all'interno delle istruzioni SQL.	no	"." (punto) "," (virgola)	(lavoro del server)
"denominazione"	Specifica la convenzione di denominazione utilizzata quando ci si riferisce a tabelle.	no	"sql" (come in schema.table) "sistema" (come in schema/tabella)	"sql"
"formato ora"	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)
"separatore ora"	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 "formato ora" 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	Necessario	Opzioni	Valore predefinito
"decimale lungo"	<p>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 decimale come viene 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.</p>	no	"true" "false"	"true"
"criteri blocco"	<p>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.</p> <p>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.</p>	no	"0" (nessun blocco record) "1" (blocca se viene specificato FOR FETCH ONLY) "2" (blocca a meno che non venga specificato FOR UPDATE)	"2"
"dimensione blocco"	<p>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 "criteri blocco" non sia non-zero. Dimensioni maggiori del blocco riducono la frequenza di comunicazioni con il server, quindi possono migliorare le prestazioni.</p>	no	"0" "8" "16" "32" "64" "128" "256" "512"	"32"



"compressione dati"	Specifica se i dati della serie di risultati vengono compressi. Se questa proprietà è impostata su "true", i dati della serie di risultati sono compressi. Se questa proprietà è impostata su "false", i dati della serie di risultati non sono compressi. La compressione dei dati può migliorare le prestazioni quando si richiamano serie di risultati estese.	no	"true" "false"	"true"
"dinamico esteso"	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. Su 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 "pacchetto".	no	"true" "false"	"false"
"chiusura lenta"	Specifica se ritardare la chiusura dei cursori fino a richieste successive. Questo aumenterà le prestazioni complessive riducendo il numero totale delle richieste.	no	"true" "false"	"false"
"limite lob"	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"	"0"

"pacchetto"	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 "dinamico esteso" non sia impostata su "true". Inoltre, è necessario impostare questa proprietà se la proprietà "dinamico esteso" è impostata su "true".	no	pacchetto SQL	""
"aggiunta pacchetto"	»Specifica se aggiungere nuove istruzioni al pacchetto SQL specificato nella proprietà "pacchetto". Questa proprietà non ha alcun effetto a meno che la proprietà "dinamico esteso" non sia impostata su "true".«	no	"true" "false"	"true"
"memoria cache del pacchetto"	»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 "dinamico esteso" non sia impostata su "true".«	no	"true" "false"	"false"
"criteri pacchetto"	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 "dinamico esteso" non sia impostata su "true".	no	"valore predefinito" (memorizza solo le istruzioni SQL con i contrassegni del parametro nel pacchetto) »"seleziona" (memorizza tutte le istruzioni SQL SELECT nel pacchetto)«	"valore predefinito"
"errore pacchetto"	Specifica come agire nel caso si verificano 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 "dinamico esteso" non sia impostata su "true".	no	"eccezione" "avvertenza" "nessuna"	"avvertenza"

"libreria pacchetto"	Specifica la libreria per il pacchetto SQL. Questa proprietà non ha alcun effetto a meno che la proprietà "dinamico esteso" non sia impostata su "true".	no	Libreria per il pacchetto SQL	"QGPL"
"corrispondenza"	Specifica se mettere in corrispondenza i dati eseguendo un'istruzione SELECT. Ciò aumenterà le prestazioni durante l'accesso alle righe iniziali in the ResultSet.	no	"true" "false"	"true"

## 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	Necessario	Opzioni	Valore predefinito
"ordinamento"	Specifica il modo in cui il server ordina i record prima di inviarli al client.	no	"hex" (basa l'ordinamento sui valori esadecimali) "lavoro" (basa l'ordinamento sull'impostazione del lavoro del server) "lingua" (basa l'ordinamento sulla serie di lingue impostata sulla proprietà "ordina lingua") "tabella" (basa l'ordinamento sulla tabella sequenza di ordinamento impostata sulla proprietà "ordina tabella")	"lavoro"
"ordina lingua"	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 "ordinamento" non venga impostata su "lingua".	no	Id lingua	ENU
"ordina tabella"	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 "ordinamento" non sia impostata su "tabella".	no	Nome tabella ordinamento completo	""
"ordina valore"	Specifica come il server considera il carattere mentre ordina i record. Questa proprietà non ha alcun effetto a meno che la proprietà "ordinamento" non sia impostata su "lingua".	no	"condiviso" (ordine dei caratteri maiuscoli e minuscoli, ad esempio lo stesso carattere) "unico" (ordine dei caratteri maiuscoli e minuscoli, ad esempio caratteri differenti)	"condiviso"

## Altre proprietà

Le altre proprietà non sono facilmente categorizzabili. 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	Necessario	Opzioni	Valore predefinito
"accesso"	Specifica il livello di accesso al database per il collegamento.	no	"tutti" (tutte le istruzioni SQL consentite) "lettura chiamata" (istruzioni SELECT e CALL consentite) "sola lettura" (solo istruzioni SELECT)	"tutti"
»"sostituzione della funzionalità"	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 sostituire la funzionalità) "1" (non eliminare 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)	""
"tipo stringa bidirezionale"	Specifica il tipo di stringa di emissione dei dati bidirezionali. Consultare <a href="#">BidiStringType</a> per ulteriori informazioni.	no	"" (utilizza il CCSID per determinare il tipo di stringa bidirezionale) "0" (il tipo di stringa predefinito per i dati non bidirezionali (LTR)) "4" "5" "6" "7" "8" "9" "10" "11"	""
"troncamento dei dati"	<p>»Specifica se il troncamento dei <b>dati carattere</b> crea messaggi di avvertenza ed eccezioni. Quando questa proprietà è "true":</p> <ul style="list-style-type: none"> <li>● Se si scrivono i dati carattere troncati sul database viene inviata un'eccezione</li> <li>● Se si utilizzano dati carattere troncati in un'interrogazione viene inviato un messaggio di avvertenza.</li> </ul> <p>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.</p> <p>Il valore predefinito è "true".</p> <p>Questa proprietà non influenza i <b>dati numerici</b>. Scrivendo dati numerici</p>	no	"true" "false"	"true"

	troncati sul database viene lanciato sempre un errore, mentre utilizzando dati numerici troncati in un'interrogazione, viene prodotto un messaggio di avvertenza.⚡			
"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 "URL secondaria".	no	"toolbox" (utilizzare solo l'unità di controllo JDBC IBM Toolbox per Java). "origine" (utilizzare l'unità di controllo JDBC IBM Developer Kit per Java se è in esecuzione sul server, altrimenti utilizzare l'unità di controllo JDBC Toolbox per Java).	"toolbox"
"errori"	Specifica il numero dei dettagli da riportare nel messaggio per gli errori che si verificano sul server.	no	"basic" "full"	"basic"
⚡"metadati estesi"	<p>Specifica se è necessario che il programma di controllo richieda metadati estesi al server. Impostare questa proprietà su True aumenta la precisione delle informazioni restituite dai seguenti metodi ResultSetMetaData:</p> <ul style="list-style-type: none"> <li>● getColumnLabel(int)</li> <li>● isReadOnly(int)</li> <li>● isSearchable(int)</li> <li>● isWritable(int)</li> </ul> <p>Inoltre, impostare questa proprietà su true abilita il supporto per il metodo ResultSetMetaData.getSchemaName(int). L'impostazione di questa proprietà su true può 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.</p> <p>E' possibile utilizzare i metadati estesi solo quando ci si collega ad un server che esegue OS/400 V5R2 o successivo.</p>	no	"true" "false"	"false"⚡

"apertura totale"	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"
"nome keyring"	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	"nome keyring"	""
"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 venga impostato su true e il nome keyring non venga impostato utilizzando la proprietà "key ring password".	no	"key ring password"	""
"server proxy"	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 <i>com.ibm.as400.access.AS400.proxyServer</i> . 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)
"annotazioni"	Specifica il sorgente del testo per le colonne REMARKS in ResultSets restituite dai metodi DatabaseMetaData.	no	"sql" (commento oggetto SQL) "sistema" (descrizione oggetto OS/400)	"sistema"
➤"salvataggio parola d'ordine quando serializzata"	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"⚡

"URL secondario"	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 dalle iSeries o dal server AS/400. 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" (codifica tutte le comunicazioni cliente/server) "false" (codificare solo la parola d'ordine)	"false"
»"traccia del server"	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 del server JDBC) "4" (avviare il debug sul lavoro del server JDBC) "8" (salvare la registrazione del lavoro quando termina il lavoro del server JDBC) "16" (avviare la traccia del lavoro sul lavoro del 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 il debug.	"0"«
"sottoprocesso utilizzato"	Specifica se è necessario utilizzare i sottoprocessi nella comunicazione con i server host.	no	"true" "false"	"true"
"trace"	Specifica se è necessario registrare i messaggi traccia. I messaggi traccia sono utili per i programmi di debug che richiamano JDBC. Tuttavia, vi è un'anomalia nelle prestazioni con la registrazione dei messaggi traccia, così questa proprietà dovrebbe essere impostata solo su "true" per il debug. I messaggi traccia vengono registrati su System.out.	no	"true" "false"	"false"
"converti binario"	Specifica se i dati binario sono convertiti. Se questa proprietà è impostata su "true", i campi BINARY e VARBINARY vengono trattati come campi CHAR e VARCHAR.	no	"true" "false"	"false"

# 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 sulle proprietà aggiuntive 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

**Esempi:** utilizzare la classe AS400JDBCBlob per leggere da un blob:

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

**Esempio:** utilizzare la 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 la modifica. In questo modo si
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 ed impostare i parametri e le colonne per nome](#), anche se l'utilizzo dell'indice delle colonne fornisce prestazioni migliori. «

Il seguente esempio mostra come utilizzare l'interfaccia CallableStatement.

```
// Collegarsi al server.
Connection c = DriverManager.getConnection("jdbc:as400://mySystem");

// Creare l'oggetto CallableStatement.
// Esso precompila l'istruzione
// 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](#) per informazioni sulle proprietà aggiuntive 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

**Esempio:** utilizzare la classe AS400JDBCClob per leggere da un clob:

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

**Esempio:** utilizzare la 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. Ciò 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 AS400JDBCCConnection

La classe AS400JDBCCConnection fornisce un collegamento JDBC ad uno specifico database UDB DB2 per iSeries. Utilizzare DriverManager.getConnection() per creare nuovi oggetti AS400JDBCCConnection. Per ulteriori informazioni, consultare [AS400JDBCDriver](#).

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](#) per una lista completa di proprietà supportate da AS400JDBCDriver.

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


AS400JDBCCConnection 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 al supporto JDBC Toolbox per Java](#).

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 AS400JDBCCConnection. L'impostazione della parola d'ordine eliminerà il contenuto di ogni ticket Kerberos o token del profilo. Per ulteriori informazioni, consultare la [Classe AS400](#) e [J2SDK, documentazione di sicurezza v1.4](#) .

Utilizzando la classe AS400JDBCCConnection, 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, è possibile utilizzare AS400JDBCCConnection 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 i 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 Toolbox 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 ad aumentare 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)<sup>(TM)</sup>. Per ulteriori informazioni sui tecnici di manutenzione JNDI, consultare [Collegamenti di riferimento IBM Toolbox per Java](#).

## Esempio: utilizzare 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 una lista 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 una lista 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 il 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 [AS400JDBC pooledConnection](#).

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 AS400JDBCDataSource. 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: creare 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: creare 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: utilizzare le 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 sul file system 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: utilizzare gli oggetti AS400JDBCDataSource e le 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"
```

-  Registrazione utilizzando il programma Java

Per caricare l'unità di controllo JDBC di 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 Toolbox per Java si registra durante il caricamento; questa è la modalità preferenziale di registrazione del programma di controllo. E' possibile anche registrare esplicitamente l'unità di controllo JDBC di 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 [Gestire i collegamenti](#) per informazioni sulla gestione dei collegamenti.

## Utilizzare l'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:** il systemName o il defaultSchema può essere omissso dall'URL.

 Per utilizzare i certificati Kerberos, impostare solo il nome del sistema (e non la parola d'ordine) sull'oggetto JDBC URL. 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 [classe AS400](#) e la

## Esempi: utilizzare l'unità di controllo JDBC per collegarsi al server

**Esempio 1:** utilizzare una URL con nome di sistema non specificato. All'utente verrà richiesto di immettere il nome del sistema con il quale si desidera collegarsi.

```
"jdbc:as400:"
```

**Esempio 2:** collegarsi al database del server; nessuno schema predefinito o proprietà specificati.

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

**Esempio 3:** collegarsi al database del server; schema predefinito specificato.

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

**Esempio 4:** collegarsi al database del server; le proprietà sono specificate utilizzando `java.util.Properties`. 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](#) per una lista 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 5:** collegarsi al database del server; le proprietà sono specificate utilizzando un'URL (uniform resource locator).

```
// Collegarsi usando 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 6:** collegarsi al database del server; l'ID utente e la parola d'ordine sono specificati.

```
// 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 7:** scollegarsi dal database. Utilizzare il metodo close() nell'oggetto Connecting per scollegarsi dal server. Per chiudere il collegamento creato nell'esempio precedente, utilizzare la seguente istruzione:

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



# 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: utilizzare 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 PreparedStatement 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 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](#).

Il seguente esempio mostra come utilizzare l'interfaccia PreparedStatement.

```
// Collegarsi al server.
Connection c = DriverManager.getConnection("jdbc:as400://mySystem");

// Creare l'oggetto PreparedStatement.
// Esso precompila l'istruzione
// SQL specificata. I punti 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();
```

# 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 ed aggiornare le colonne per nome](#), anche se l'utilizzo dell'indice 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	
<a href="#">absolute</a>	<a href="#">isFirst</a>
<a href="#">afterLast</a>	<a href="#">isLast</a>
<a href="#">beforeFirst</a>	<a href="#">last</a>
<a href="#">first</a>	<a href="#">moveToCurrentRow</a>
<a href="#">getRow</a>	<a href="#">moveToInsertRow</a>
<a href="#">isAfterLast</a>	<a href="#">previous</a>
<a href="#">isBeforeFirst</a>	<a href="#">relative</a>

## 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 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 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 alle modifiche effettuate quando è aperta, mentre la serie di risultati sensibile allo scorrimento è sensibile alle modifiche.»L'unità di controllo JDBC IBM Toolbox per Java non supporta serie di risultati insensibili 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. Numerosi metodi di aggiornamento sono disponibili, ad esempio:

- [Aggiornare il flusso ASCII](#)
- [Aggiornare il Big Decimal](#)
- [Aggiornare il flusso binario](#)

Consultare [Riepilogo metodi](#) per una lista completa dei 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 il tipo di serie di
risultati
// aggiornabile simultaneamente.
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 che esegue OS/400 V5R2 o versioni successive, l'utilizzo della [proprietà metadati estesi](#) consente di aumentare l'accuratezza dei seguenti metodi ResultSetMetaData:

- getColumnLabel(int)
- isReadOnly(int)
- isSearchable(int)
- isWritable(int)

Inoltre, impostare questa proprietà su true abilita il supporto per il metodo ResultSetMetaData.getSchemaName(int). Tenere presente l'utilizzo della proprietà metadati estesi può rallentare le prestazioni poiché richiede il richiamo di più informazioni dal server. ⚡

# AS400JDBCRowSet

La classe [AS400JDBCRowSet](#) rappresenta un rowset collegato che incorpora una serie di risultati JDBC. I metodi su AS400JDBCRowSet sono molto simili a quelli di [AS400JDBCResultSet](#). Il collegamento database è mantenuto durante l'utilizzo.

E' possibile utilizzare un oggetto [AS400JDBCDataSource](#) o un oggetto [AS400JDBCConnectionPoolDataSource](#) per creare il collegamento al database che si vuole utilizzare per accedere ai dati per di AS400JDBCRowSet.

## Esempi

I seguenti esempi mostrano come è possibile utilizzare la classe AS400JDBCRowSet.

### Esempio: creare, popolare ed aggiornare 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 la lista.
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: creare e popolare un oggetto AS400JDBCRowSet, mentre viene richiamato 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 nel 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: utilizzare i punti di salvataggio

In questo scenario, supponiamo 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
```

```

intermediato 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

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, 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 rimangono. 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 Rilascio 2 o versioni successive. L'utilizzo dei punti di salvataggio quando si effettua un collegamento (o si è già collegati) ad un server che esegue V5R1 o versioni precedenti di OS/400 invia un'eccezione. ⚡

# 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:

- [Statement](#) - per eseguire una semplice istruzione SQL che non possiede parametri.
- [PreparedStatement](#) - per eseguire un'istruzione precompilata SQL che può possedere o meno parametri.
- [CallableStatement](#) - per eseguire una chiamata ad 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 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](#).

Quando si utilizzano aggiornamenti batch, generalmente 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 una lista di comandi che possono essere inoltrati con esito positivo ed eseguiti insieme in un gruppo. Quando questa lista di comandi batch viene eseguita dal metodo executeBatch(), i comandi vengono eseguiti nell'ordine nel quale essi sono stati aggiunti alla lista.

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, inseriti nel bundle Java 2 Platform, Standard Edition, versione 1.4.«

Per ulteriori informazioni, consultare i siti Web di 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](#).
- [AS400JDBCXAConnection](#) - 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: utilizzare le 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 una volta terminato. Questa operazione
chiude
    // esplicitamente la risorsa XA.
xaConnection.close();
```

# Le classi dei lavori

Le classi Jobs di IBM Toolbox per Java (nel pacchetto Access) consentono al programma Java di richiamare e modificare le informazioni sul lavoro.

**Nota:** Toolbox per Java fornisce anche [classi resource](#) che presentano una framework generica e un'interfaccia di programmazione coerente per gestire i vari oggetti e le varie liste 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 una lista di lavori iSeries
- [JobLog](#) - rappresenta la registrazione lavori di un iSeries

## Esempi

Elencare i lavori appartenenti ad un [utente specifico](#) ed elencare i lavori con [le informazioni relative allo stato del lavoro](#).

Visualizzare i messaggi in una [registrazione lavori](#).

Utilizzare una memoria cache durante l'impostazione 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");
            // Convalidare le modifiche. Ciò modificherà il valore sull'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("error : " + e)  
}
```

# Job

La [classe Job](#) (nel pacchetto di accesso) consente ad un programma java di richiamare e modificare le informazioni sul lavoro del server.

**Nota:** Toolbox per Java fornisce anche [classi resource](#) che presentano una framework generica e un'interfaccia di programmazione coerente per gestire i vari oggetti e le varie liste 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 metodo [setCacheChanges\(true\)](#) ed eseguire il commit delle modifiche utilizzando il metodo [commitChanges\(\)](#). Se la memorizzazione in cache non è attiva, non è necessario eseguire il commit.

Utilizzare questo [esempio](#) per comprendere 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\(\)](#).

# Classe JobList

E' possibile utilizzare la classe [JobList](#) (nel pacchetto Access) per elencare i [lavori](#) iSeries.

**Nota:** Toolbox per Java fornisce anche [classi resource](#) che presentano una framework generica e un'interfaccia di programmazione coerente per gestire i vari oggetti e le varie liste 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 restituire una lista di lavori iSeries o il metodo [getLength\(\)](#) per restituire il numero di lavori richiamati con l'ultimo [getJobs\(\)](#).

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 lista lavori.
JobList jobList = new JobList(sys);

        // Richiamare la lista dei 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());
}
```

# JobLog

La classe [JobLog](#) (nel pacchetto Access) richiama i messaggi nella registrazione lavori di un lavoro server tramite la chiamata a [getMessages\(\)](#).

**Nota:** Toolbox per Java fornisce anche [classi resource](#) che presentano una framework generica e un'interfaccia di programmazione coerente per gestire i vari oggetti e le varie liste 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](#).

Il seguente esempio stampa tutti i messaggi nella registrazione lavori per l'utente specificato:

```
AS400                                     // ... Impostare il lavoro per creare un oggetto
                                           // mentre un oggetto jobList è già stato
                                           // impostato

                                           // Richiamare la lista dei lavori attivi
                                           // su iSeries
Enumeration list = jobList.getJobs();

                                           // Consultare la lista 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 iSeries che viene 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 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:

```
                                // 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

I seguenti esempi mostrano come è possibile utilizzare liste di messaggi con CommandCall e ProgramCall.

- **Esempio:** [utilizzare una lista di messaggi con CommandCall](#)
- **Esempio:** [utilizzare una lista di messaggi con ProgramCall](#)

## QueuedMessage

La [classe QueuedMessage](#) estende la classe AS400Message.

**Nota:** Toolbox per Java fornisce anche [classi resource](#) che presentano una framework generica e un'interfaccia di programmazione coerente per gestire i vari oggetti e le varie liste 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 i messaggi in coda è [RQueuedMessage](#).

La classe QueuedMessage accede ad 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):

```

// 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 la lista dei 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 il [testo di sostituzione](#) nel messaggio

Il seguente esempio mostra come richiamare e stampare un messaggio:

```

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 ad un programma Java di interagire con una coda messaggi iSeries.

**Nota:** Toolbox per Java fornisce anche [classi resource](#) che presentano una framework generica e un'interfaccia di programmazione coerente per gestire i vari oggetti e le varie liste 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 una lista 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:

```
        // 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 la lista dei 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 una lista di tutte le [condivisioni di file](#) e [condivisioni di stampa](#)
- Richiamare una lista 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 un profilo utente server che disponga dell'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 sospenso

Molti degli attributi di NetServer sono in sospenso (ad esempio, [NAME\\_PENDING](#)). Gli attributi in sospenso 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 sospenso mentre l'altro non lo è:

- L'attributo in sospenso è lettura/scrittura, quindi è possibile modificarlo
- L'attributo non in sospenso 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 del 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

```
// 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");
// Convalidare 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 la lista 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 la lista delle autorizzazioni relative all'oggetto.
- [setSensitivityLevel\(\)](#): imposta il livello di sensibilità dell'oggetto.

## Esempio

Questo 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
<a href="#">DLOPermission</a>	Rappresenta l'autorizzazione di un utente ai DLO (Document Library Objects), memorizzati in QDLS.

<a href="#">QSYSPermission</a>	Rappresenta l'autorizzazione di un utente agli oggetti memorizzati in QSYS.LIB e contenuti nel server.
<a href="#">RootPermission</a>	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 della lista di autorizzazioni

## Esempio

Questo esempio mostra come richiamare gli utenti e i gruppi autorizzati ad un oggetto e come stamparli l'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 liste di autorizzazioni.
*AUTL	La lista di autorizzazioni è utilizzata per determinare l'autorizzazione al documento.
*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 Chapter 5: Resource Security in the [iSeries](#)

[Security Reference](#) 

## Esempio

Questo esempio mostra come richiamare e stampare le autorizzazioni DLO, inclusi i profili utente per ogni permesso.

```
// 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 dispone un utente per un oggetto contenuto nella struttura della libreria tradizionale di iSeries o di AS/400e memorizzato 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 liste di autorizzazioni.
*AUTL	La lista di autorizzazioni è utilizzata per determinare l'autorizzazione al documento.
*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:

Autorizzazione definita dal sistema	Autorizzazione per gli oggetti					Autorizzazione per i dati				
	Opr	Mgt	Exist	Alter	Ref	Read	Add	Upd	Dlt	Exe
<b>Tutto</b>	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
<b>Change</b>	Y	n	n	n	n	Y	Y	Y	Y	Y
<b>Exclude</b>	n	n	n	n	n	n	n	n	n	n
<b>Use</b>	Y	n	n	n	n	Y	n	n	n	Y
<b>Autl</b>	Valido solo per utente (*PUBLIC) e per una lista autorizzazioni specifica che determina le singole autorizzazioni per gli oggetti e per i dati.									

Y fa riferimento alle autorizzazioni che è possibile assegnare.  
n fa riferimento alle autorizzazioni che non è possibile assegnare.

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 Chapter 5: Resource security in [iSeries Security Reference](#) . 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 dalla lista 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 Chapter 5: Resource security in [iSeries](#)

[Security Reference](#) 

## 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 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
- [Visualizzazione](#) dei file di spool di AFP (Advanced Function Printing) e SCS (SNA Character Stream)

## Esempi

- L'[Esempio Creare il file di spool](#) mostra come creare un file di spool su un server da un flusso di immissioni.
- L'[Esempio Creare file di spool SCS](#) mostra come creare un flusso di dati SCS utilizzando la classe SCS3812Writer e come scrivere il flusso sul file di spool sul server.
- L'[Esempio Leggere il file di spool](#) mostra come leggere un esistente file di spool del server.
- Il primo [Esempio di lista asincrona](#) mostra come elencare in modo asincrono tutti i file di spool su un sistema e come utilizzare l'interfaccia PrintObjectListListener per richiamare il feedback di creazione della lista.
- Il secondo [Esempio lista asincrona](#) mostra come elencare in modo asincrono tutti i file di spool su un sistema *senza* utilizzare l'interfaccia PrintObjectListListener
- L'[Esempio lista sincrona](#) mostra come elencare in modo asincrono tutti i file di spool sul sistema.

# Elencare gli oggetti di stampa

E' possibile utilizzare la classe [PrintObjectList](#) e le sue sottoclassi per gestire le liste degli oggetti di stampa. Ogni sottoclasse contiene metodi che consentono di filtrare la lista in base a ciò che risulta appropriato per quel particolare tipo di oggetto di stampa. Ad esempio, [SpooledFileList](#) consente di filtrare una lista 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 la lista 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 nella lista non vengono richiamati dal server. Il metodo [openAsynchronously\(\)](#) viene restituito immediatamente e il chiamante può effettuare altre operazioni in primo piano mentre attende la creazione della lista. La lista aperta in modo asincrono consente anche al chiamante 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 nella lista.

Se la lista viene aperta in modo asincrono, il chiamante può ottenere un feedback sulla creazione della lista. I metodi, come [isCompleted\(\)](#) e [size\(\)](#), indicano se la creazione della lista è terminata o restituiscono la dimensione corrente della lista. Altri metodi, [waitForListToComplete\(\)](#) e [waitForItem\(\)](#), consentono al chiamante di attendere che la lista sia completa oppure di attendere una voce particolare. Inoltre, per richiamare questi metodi [PrintObjectList](#), il chiamante si può registrare con la lista come listener. In questa situazione, il chiamante è informato sugli eventi che si verificano nella lista. Per registrare o per annullare la registrazione degli eventi, il chiamante 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 <a href="#">PrintObjectList</a>	Quando l'evento è distribuito
<a href="#">listClosed</a>	Quando la lista viene chiusa.
<a href="#">listCompleted</a>	Quando la lista è completa.
<a href="#">listErrorOccurred</a>	Se viene inviata un'eccezione quando la lista viene richiamata.
<a href="#">listOpened</a>	Quando la lista è aperta.
<a href="#">listObjectAdded</a>	Quando un oggetto viene aggiunto alla lista.

Una volta aperta la lista e una volta elaborati gli oggetti della lista, chiudere la lista 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 chiusa la lista, i filtri possono essere modificati e la lista stessa può essere riaperta.

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 una lista predefinita di attributi per ogni tipo di oggetto di stampa che può essere sostituita utilizzando il metodo [setAttributesToRetrieve\(\)](#) in [PrintObjectList](#). Consultare la sezione [Richiamare attributi \[PrintObject\]\(#\)](#) per una lista degli attributi supportati da ogni tipo di oggetto di stampa.

La lista delle risorse AFP è consentita solo nella Versione 3 Rilascio 7 e successivi rilasci di OS/400. L'apertura di un [AFPResourceList](#) su un sistema precedente a V3R7 crea un'eccezione [RequestNotSupportedException](#).

## Esempi

[Esempio 1 di lista asincrona](#)

[Esempio 2 di lista asincrona](#)

[Esempio di lista sincrona](#)

## Gestire gli 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 una lista degli oggetti, quindi, richiamare i singoli oggetti tramite la lista.
- 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](#)

## Richiamare 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.
- Utilizzare [getFloatAttribute\(int attributeID\)](#) per richiamare un attributo di tipo floating point.
- Utilizzare [getStringAttribute\(int attributeID\)](#) 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 una lista 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 una lista 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. La lista in avvio è vuota ed il chiamante può aggiungere gli attributi alla lista 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 una lista 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 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
- [ATTR\\_BKMGN\\_ACR](#) - Scostamento trasversale margine posteriore
- [ATTR\\_BKMGN\\_DWN](#) - Scostamento verso il basso margine posteriore
- [ATTR\\_BACK\\_OVERLAY](#) - Nome IFS sovrapposizione lato 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 delle pagina
- [ATTR\\_EDGESTITCH\\_NUMSTAPLES](#) - Numero di punti graffettatura bordo foglio
- [ATTR\\_EDGESTITCH\\_REF](#) - Riferimento graffettatura a bordo foglio
- [ATTR\\_EDGESTITCH\\_REFOFF](#) - Scostamento riferimento graffettatura 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 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\\_PAGELN](#) - Lunghezza della 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 graffettatura centrale dei punti
- [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 oggetto definito dall'utente

## Impostare attributi

E' possibile utilizzare i seguenti attributi per un file di stampa utilizzando il metodo setAttributes():

- [ATTR\\_ALIGN](#) - Allineare pagina
- [ATTR\\_BKMGN\\_ACR](#) - Scostamento trasversale margine posteriore
- [ATTR\\_BKMGN\\_DWN](#) - Scostamento verso il basso margine posteriore
- [ATTR\\_BACK\\_OVERLAY](#) - Nome IFS sovrapposizione lato 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 delle pagina
- [ATTR\\_EDGESTITCH\\_NUMSTAPLES](#) - Numero di punti graffettatura bordo foglio
- [ATTR\\_EDGESTITCH\\_REF](#) - Riferimento graffettatura a bordo foglio
- [ATTR\\_EDGESTITCH\\_REFOFF](#) - Scostamento riferimento graffettatura 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 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\\_PAGELN](#) - Lunghezza della 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 graffettatura centrale dei punti
- [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 oggetto definito dall'utente

# Attributi file di spool

---

## Come richiamare gli 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 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\\_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\\_PAGRIT - 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 - Nome utente che crea lavoro IPP](#)
- [ATTR\\_IPP\\_JOB\\_ORIGUSER\\_NL - NL nome utente che crea lavoro IPP](#)
- [ATTR\\_IPP\\_PRINTER\\_NAME - Nome stampante IPP](#)
- [ATTR\\_JOBNAME - Nome lavoro](#)
- [ATTR\\_JOBNUMBER - Numero lavoro](#)
- [ATTR\\_JOBUSER - 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\\_PAGELN - Lunghezza pagina](#)
- [ATTR\\_PAGEWIDTH - Ampiezza pagina](#)
- [ATTR\\_MEASMETHOD - Metodo di misurazione](#)
- [ATTR\\_NETWORK - Identificativo di rete](#)
- [ATTR\\_NUMBYTES - Numero di byte in lettura/scrittura](#)
- [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](#) - Riduzione emissione
  - [ATTR\\_RPLUNPRT](#) - Sostituire caratteri non stampabili
  - [ATTR\\_RPLCHAR](#) - Carattere di sostituzione
  - [ATTR\\_RESTART](#) - Riavviare stampa
  - [ATTR\\_SADDLESTITCH\\_NUMSTAPLES](#) - Numero graffettatura centrale dei punti
  - [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
-

# Impostare 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 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
-

## Creare 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, `QPNPSRPTF`. 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 una lista completa 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 dovrebbe utilizzarlo generalmente, a meno che non si verifichi il più delle volte un caso specifico in cui l'immissione automatica dei dati non funziona. 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 HP-PCL e postscript sono esempi di flussi di dati che dovrebbero trovarsi in un file di spool \*USERASCII.

## Esempi

[Esempio di creazione del file di spool](#)

[Esempio di creazione del file di spool SCS](#)

# Creare un flusso di dati SCS

Per creare file di spool che si utilizzeranno su determinate stampanti collegate al server, deve essere creato 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 chiamante dovrebbe scegliere il programma di scrittura corrispondente alla destinazione della stampante finale sulla quale il chiamante o l'utente finale eseguiranno 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
<a href="#">SCS5256Writer</a>	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.
<a href="#">SCS5224Writer</a>	Estende il programma di scrittura 5256 e aggiunge i metodi per impostare CPI (character per inch) e LPI (lines per inch).
<a href="#">SCS5219Writer</a>	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.
<a href="#">SCS5553Writer</a>	Estende il programma di scrittura 5219 ed aggiunge il supporto per la rotazione caratteri, righe griglia e adattamento font. Il 5553 è un flusso di dati DBCS (double byte character set).
<a href="#">SCS3812Writer</a>	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 chiamante 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 chiamante 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.

## Leggere i file di spool e le risorse AFP

E' possibile utilizzare la classe [PrintObjectInputStream](#) 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 Rilascio (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 immissioni. 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 chiamante può utilizzare i metodi [mark\(\)](#) e [reset\(\)](#) per spostare la posizione attuale di lettura indietro nel flusso di immissioni. Utilizzare il metodo [skip\(\)](#) per spostare la posizione attuale di lettura in avanti nel flusso di immissioni senza leggere i dati.

## Esempio

[Esempio lettura file di spool](#)

# Leggere i 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](#).

# 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 significa che 15 utenti possono utilizzare il prodotto contemporaneamente, ma non devono essere 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 una lista di oggetti [AS400Message](#) .

I parametri richiesti sono i seguenti:

- Il programma ed i parametri da eseguire
- L'oggetto [AS400](#) che rappresenta il sistema iSeries su cui è installato il programma.

Il nome programma e la lista 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.

La classe dell'oggetto ProgramCall fa sì che l'oggetto AS400 si colleghi ad iSeries.

Il seguente esempio mostra come utilizzare la classe ProgramCall:

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

// Creare un oggetto programma. Si è scelto di
impostare // il programma per essere eseguito 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 proprio programma non
// ha parametri. Se l'esecuzione non riesce, l'errore
// viene restituito come una serie di messaggi
// nella lista messaggi.
if (pgm.run() != true)
{
    // Se si è giunti a questo punto, l'esecuzione del
    // programma ha avuto esito negativo. Richiamare la
    // lista dei messaggi per determinare il motivo per
    // cui il programma non è stato eseguito.
    AS400Message[] messageList = pgm.getMessageList();

    // ... Elaborare la lista dei messaggi.
}

// Scollegarsi in quanto è terminata
// l'esecuzione dei programmi
sys.disconnectService(AS400.COMMAND);
```

L'oggetto ProgramCall richiede il [nome percorso IFS](#) del programma.

L'utilizzo della classe ProgramCall fa sì che l'oggetto AS400 si colleghi ad iSeries. Consultare [Gestire i collegamenti](#) per informazioni sulla gestione dei collegamenti.

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\(\)](#).

## Utilizzare gli 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 una lista.

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 di
        // quattro byte.
parmList[1] = new ProgramParameter(4);

        // Creare un oggetto programma
        // specificando il nome del
        // programma e la lista del parametro.
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 la lista dei messaggi per
        // individuare il motivo per cui non è stato
eseguito.
        AS400Message[] messageList = pgm.getMessageList();

}

        else
{

        // Altrimenti è stato eseguito il programma.
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.
        // Il numero parte dall'inizio del
        // beginning of the 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.

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 viene richiamato interamente. 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 [gestire i 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.

## Specificare la 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. Questa operazione
    // deve essere eseguita 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)
{ // Found the record for John Doe, print out the info.
    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. Questa operazione
// deve essere eseguita 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);
```

```
in quanto // Utilizzare il metodo update() della classe di base
// 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.

### Creare il codice sorgente Java per le sottoclassi di RecordFormat per rappresentare il formato record dei 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 quando vengono creati i file sorgente Java.

**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);
```

### Creare gli 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);
```

# 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 sì che l'oggetto AS400 si colleghi a iSeries. Consultare [gestire i 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 programmi di servizio siano eseguiti nel lavoro Java utilizzando il metodo ereditato (da ProgramCall) [setThreadSafe\(\)](#).

## Utilizzare la classe ServiceProgramCall

Per utilizzare la classe ServiceProgramCall, è necessario assicurarsi di soddisfare i requisiti che seguono:

- Il programma di servizio si deve trovare su un server iSeries o AS/400e su cui è in esecuzione OS/400 V4R4 o successive
- 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

## Gestire gli 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

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();
```



# 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](#)
- [Rilascio](#)
- [Valore](#)

Utilizzando la classe SystemValue, richiamare un valore di sistema singolo utilizzando il metodo [getValue\(\)](#) e modificare il valore di sistema utilizzando il metodo [setValue\(\)](#).

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 si desidera il valore attuale iSeries invece del valore memorizzato nella cache, è necessario utilizzare il metodo [clear\(\)](#) per eliminare il contenuto della cache attuale.

## SystemValueList

[SystemValueList](#) rappresenta una lista dei valori di sistema sul sistema iSeries specificato. La lista è divisa in diversi [gruppi definiti dal sistema](#) che consentono al programma Java di accedere ad una parte dei valori di sistema alla volta.

## SystemValueGroup

[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 che SystemValueGroup viene popolato con i nomi valore di sistema desiderati, ottenere gli oggetti SystemValue reali dal gruppo richiamando 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 come utilizzare le classi SystemValue e SystemValueList

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 una lista 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 come utilizzare la 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");

//Ottenerne 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 i valori di sistema derivano da diversi sistemi iSeries.
//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();
```

# Trace

L'oggetto [Trace](#) consente al programma Java di registrare punti di traccia e messaggi diagnostici. Queste informazioni aiutano a riprodurre e diagnosticare i problemi

**Nota:** è possibile 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 deve essere utilizzare solo dalle classi IBM Toolbox per Java.
Flusso di dati	Registra i dati che si trovano nei programmi iSeries e Java. Questa categoria deve essere utilizzare 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 utilizzare per determinare come PCML interpreta i dati 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.
Tutto	Questa categoria viene utilizzare per abilitare o disabilitare l'esecuzione della traccia di tutte le categorie in una volta. Le informazioni di traccia non possono essere direttamente registrate 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 componente, 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 determinare se creare il record traccia prima che richiami 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 metodo [setFileName\(\)](#) 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.** I programmi Java devono fornire all'utente un metodo per attivare o registrare in modo che sia facile abilitare la registrazione. Ad esempio, l'applicazione può analizzare un parametro della riga comandi che indica quale categoria di dati deve essere registrata. L'utente può impostare questo parametro quando sono necessarie informazioni di registrazione.

Gli esempi che seguono mostrano come utilizzare la classe Trace

**Esempio 1:** quanto segue è un esempio di come utilizzare il metodo `setTraceOn` e di come scrivere i dati in una registrazione utilizzando il metodo di 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 2:** l'esempio che segue mostra come utilizzare la traccia. Il metodo 2 è il modo migliore per scrivere il codice che utilizza la traccia.

```
// 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 3:** l'esempio che segue mostra come utilizzare la traccia del componente.

```
// Creare una stringa componente. E' più efficiente per creare un
// oggetto rispetto a molte costanti letterali String.
String myComponent1 = "com.myCompany.xyzComponent";
String myComponent2 = "com.myCompany.abcComponent";

// Inviare Toolbox e i dati traccia componente per separare i file.
// La traccia Toolbox 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 traccia specifici del componente o i dati di
// traccia toolbox generali.

Trace.setFileName("c:\\bit.bucket");
Trace.setFileName(myComponent1, "c:\\Component1.log");
```

Ciò che segue è il risultato dell'esempio, se non si specificano file di traccia:

```
Toolbox for Java - Version 5 Release 1 Modification level 0
[com.myCompany.xyzComponent] Tue Oct 24 16:02:44 CDT 2000 I am here
[com.myCompany.abcComponent] Tue Oct 24 16:02:44 CDT 2000 I am there
Tue Oct 24 16:02:44 CDT 2000 I am everywhere
```

# Users e Groups

Le classi Users e Groups consentono di richiamare una lista di utenti e di gruppi utenti sul sistema iSeries nonché le informazioni riguardo ogni utente tramite un programma Java.

**Nota:** Toolbox per Java fornisce anche [classi resource](#) che presentano una framework generica e un'interfaccia di programmazione coerente per gestire i vari oggetti e le varie liste 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](#), è necessario utilizzare il metodo [setSystem\(\)](#) per impostare il nome del sistema e 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 restituita una lista di utenti membri del gruppo.

Il programma Java può scorrere la lista 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 la lista di tutti gli utenti e gruppi.
    Enumeration enum = userList.getUsers ();

// Iterare la lista.
    while (enum.hasMoreElements ())
    {
        User u = (User) enum.nextElement ();
        System.out.println (u);
    }
```

## Richiamo di informazioni sugli utenti e sui gruppi

Utilizzare uno [UserList](#) per richiamare una lista 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 la lista di utenti deve essere richiamata.

Per impostazione predefinita, vengono restituiti tutti gli utenti. Utilizzare una combinazione di [setUserInfo\(\)](#) e [setGroupInfo\(\)](#) per specificare esattamente quali utenti devono essere restituiti.

**Esempio:** utilizzare UserList per elencare tutti gli utenti di un determinato 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.
- Impostare gli attributi di uno spazio utente. Un programma Java può impostare il [valore iniziale](#), il [valore lunghezza](#) e gli attributi [estensibili 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 [gestire i collegamenti](#) per informazioni sulla gestione dei collegamenti.

L'esempio seguente crea un'area utente, quindi scrive i dati in essa.

```
// 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
// server.
US.create(10240, // La dimensione iniziale è
10K 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 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
- Modificare il linguaggio e la direzione del testo
- Creare liste ordinate e non ordinate
- Creare liste 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.

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 [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 [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 tag immagine per le pagine HTML. ⚡
- Le classi [HTMLList](#) aiutano a creare delle liste 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 sia la classe HTML che 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 relative all'utilizzo della tag HTML <BDO>, consultare il sito web [W3C](#) .

## Esempio: utilizzare BidiOrdering

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> ed 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 dalla lista 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 dell'elemento di immissione
- [Richiamare una rappresentazione di stringa](#) dell'oggetto HTMLAlign

## Esempio: creare oggetti HTMLAlign

L'esempio che segue crea una lista disordinata, quindi crea un oggetto HTMLAlign per allinearla:

```
// Creare una lista non ordinata.
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 la lista.
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:

- Lista non ordinata centrata
- Un altro elemento

# 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, la lista dei 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 nella lista 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 tag che segue:

```
<input type="hidden" name="account" value="123456" />
```

In una pagina HTML, HiddenFormInput non viene visualizzato. Invia semplicemente le informazioni al server (in questo caso il numero di conto).

# 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](#) o [impostare](#) l'allineamento
- [Richiamare](#) o [impostare](#) l'altezza
- [Richiamare](#) o [impostare](#) l'ampiezza

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 inoltra 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 `TextFormField`

La classe [`TextFormField`](#) rappresenta un tipo di immissione di testo a riga singola in formato HTML. La classe `TextFormField` 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 `TextFormField`:

```
TextFormField text = new TextFormField("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.
- [LinearLayoutFormPanel](#): rappresenta un layout riga degli 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 `LinearLayoutFormPanel`

La classe [`LinearLayoutFormPanel`](#) 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.

Questo esempio crea un oggetto `LinearLayoutFormPanel` 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);
LinearLayoutFormPanel panel = new LinearLayoutFormPanel();
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 determina la dimensione dell'area testo impostando il numero di [righe](#) e di [colonne](#). E' possibile determinare la dimensione impostata per un elemento dell'area testo con i metodi [getRows\(\)](#) e [getColumns\(\)](#).

Il testo iniziale nell'area testo viene impostato con il metodo [setText\(\)](#). Si utilizza il metodo [getText\(\)](#) 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.

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' possibile [aggiungere](#) 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:

- Utilizzare [setMultiple\(\)](#) 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 un elemento del formato di opzione 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 il valore predefinito dell'opzione è impostato su [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 aggiungeli.
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 HTMLHeading

La classe [HTMLHeading](#) rappresenta un'intestazione HTML. Ogni intestazione ha 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 dell'elemento immesso
- [Richiamare una rappresentazione di stringa](#) dell'oggetto HTMLHeader

## Esempio: creare 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");

// 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>
```

Quando si utilizza questa tag in una pagina HTML, essa viene visualizzata in questo modo:

[Home page di IBM Toolbox per Java](http://www.ibm.com/as400/toolbox)

## » 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.

```

```



# Classi HTMLList

Le classi HTMLList consentono di creare facilmente le liste all'interno delle pagine HTML. Queste classi forniscono metodi per richiamare e impostare vari attributi delle liste e le voci in esse contenute.

In particolare, la classe principale [HTMLList](#) fornisce un metodo per produrre una [lista compatta](#) che visualizza le voci nel minor spazio verticale possibile.

- I metodi per [HTMLList](#) includono:
  - [Comprimere](#) la lista
  - [Aggiungere](#) e [rimuovere](#) voci dalla lista
  - [Aggiungere](#) e [rimuovere](#) liste dalla lista (rendendo possibile la nidificazione delle liste stesse)
- 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 le liste HTML:

- [OrderedList](#) e [OrderedListItem](#)
- [UnorderedList](#) e [UnorderedListItem](#)

Per gli snippet di codifica, consultare gli esempi che seguono:

- **Esempio:** [creare liste ordinate](#)
- **Esempio:** [creare liste non ordinate](#)
- **Esempio:** [creare liste nidificate](#)

## OrderedList e OrderedListItem

Utilizzare le classi [OrderedList](#) e [OrderedListItem](#) per creare liste ordinate nelle pagine HTML.

- I metodi per OrderedList includono:
  - [Richiamare](#) e [impostare](#) il numero iniziale per la prima voce nella lista
  - [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 nella lista.

Consultare l'esempio per [creare liste ordinate](#).

## UnorderedList e UnorderedListItem

Utilizzare le classi [UnorderedList](#) e [UnorderedListItem](#) per creare liste non ordinate 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 liste non ordinate](#).

## Esempi

Gli esempi che seguono mostrano come utilizzare classi `HTMLList` per creare liste ordinate, non ordinate e nidificate.

### Esempio: creare liste ordinate

L'esempio che segue crea una lista ordinata:

```
// 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 della lista 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>
```

Quando si utilizzano queste tag in una pagina HTML, essa viene visualizzata in questo modo:

- i. Primo elemento
- ii. Secondo elemento

### Esempio: creare liste non ordinate

L'esempio che segue crea una lista non ordinata:

```
// 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 lista 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>

```

Quando si utilizzano queste tag in una pagina HTML, essa viene visualizzata in questo modo:

- Primo elemento
- Secondo elemento

## Esempio: creare liste nidificate

L'esempio che segue crea una lista nidificata:

```

// 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 lista 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 della lista all'OrderedList.
oList.addListItem(listItem1);
oList.addListItem(listItem2);
// Aggiungere (nidificare) la lista non ordinata 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: creare 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 metal = new HTMLMeta();
metal.setName("keywords");
metal.setLang("en-us");
metal.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(metal + "\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: creare 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: creare 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 una [riga](#) 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

Esempio: [utilizzare le classi HTMLTable](#).

# Classe HTMLTableCell

La classe [HTMLTableCell](#) prende qualsiasi oggetto [HTMLTagElement](#) 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
- [Impostare](#) se i dati della cella utilizzeranno o meno le normali convenzioni di interruzione di 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"><a href="http://www.ibm.com">IBM Home Page</a></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 specifico di cella, la cella di intestazione, 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 della lista di file
- [FileListRenderer](#): restituisce la lista di file e indirizzari

## Esempi

Gli esempi che seguono mostrano differenti metodi di utilizzo delle classi HTMLTree.

- **Esempio:** [utilizzare le classi HTMLTree](#)
- **Esempio:** [creare una gerarchia ad albero IFS](#)

# 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
- [Impostare](#) se l'elemento ad albero verrà espanso o meno

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>My Web Page</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>Another Web Page</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. »E' inoltre possibile richiamare ed impostare il nome ed 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 o meno (metodo ereditato)
- »[Richiamare](#) o [impostare](#) il nome dell'unità condivisa NetServer«
- »[Richiamare](#) o [impostare](#) il percorso dell'unità condivisa NetServer«

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 della lista 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 della lista dei file
- [Richiamare](#) e [impostare](#) la richiesta servlet HTTP
- [Richiamare](#) e [impostare](#) FileListRenderer
- [Richiamare](#) e [impostare](#) HTMLTable con cui visualizzare la lista dei file
- »[Richiamare](#) o [impostare](#) il nome di un'unità condivisa NetServer◀
- »[Richiamare](#) o [impostare](#) il percorso di una 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 della lista dei file
- Con la classe [FileTreeElement](#), è possibile creare una lista di file gerarchica IFS » o file condivisi NetServer◀

Il [javadoc FileListElement](#) mostra come creare e visualizzare un oggetto FileListElement.

## Esempio

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 NetServer.◀

- **Esempio:** [creare una gerarchia ad albero IFS](#)

## » 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)<sup>(TM)</sup>. 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 contengono una varietà di classi che consentono di formattare i flussi di dati XML e creare prospetti nei suddetti formati. Assicurarsi di avere i necessari file jar nel CLASSPATH. Per maggiori informazioni sui file jar di `reportwriter`, consultare [File jar](#).«

[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 deve mai richiamare direttamente alcuno dei metodi in ogni Classe di contesto. 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: utilizzare XSLReportProcessor con PCLContext](#)

[Esempio: utilizzare JSPReportProcessor con PDFContext](#)

# Classe JSPReportProcessor

La classe [JSPReportProcessor](#) consente di creare un documento o un prospetto del contenuto di una JSP (JavaServer Page<sup>TM</sup>) o servlet Java.

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' possibile anche [scaricare un file zip](#) che contiene i file sorgente di esempio di JSP, XML e XSL per entrambi gli esempi JSPReportProcessor e XSLReportProcessor: ⚡

- [Esempio: utilizzare JSPReportProcessor con PDFContext](#)
- ➤ Esempio: file JSP di esempio JSPReportProcessor ⚡

Per ulteriori informazioni sulle JSP, consultare [Java Server Pages technology](#)  sul [sito web Java di Sun](#) .



# 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 [XSL foglio di stile](#)
- Impostare [Sorgente dati XML](#)
- Impostare [Sorgente FO XSL](#)
- Elaborare [un prospetto](#)

Il seguente esempio mostra 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' anche possibile [scaricare un file zip](#) che contiene i file sorgente di esempio XML, XSL e JSP sia per gli esempi XSLReportProcessor che per gli esempi JSPReportProcessor: «

- [Esempio: utilizzo di XSLReportProcessor con PCLContext](#)
- »Esempio: [file esempio XML](#) XSLReportProcessor «
- »Esempio: [file esempio XSL](#) XSLReportProcessor «

Per ulteriori informazioni su XML e XSL, consultare l'argomento [XML](#) nell'Information Center.

# Classi risorsa

Il pacchetto [com.ibm.as400.resource](#) fornisce una framework generica per gestire i vari oggetti e le liste AS400. Tale framework fornisce un'interfaccia di programmazione coerente con tali oggetti e liste.

Il pacchetto risorsa 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 una lista di risorse iSeries, ad esempio una lista 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, liste 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

# Liste di risorse

La classe [com.ibm.as400.resource.ResourceList](#) rappresenta una lista di risorse iSeries. E' una classe astratta che fornisce un accesso generico al contenuto della lista.

IBM Toolbox per Java fornisce le seguenti liste di risorse:

- [RIFSFileList](#) - rappresenta una lista di file e indirizzari nell'IFS di iSeries
- [RJobList](#) - rappresenta una lista di lavori iSeries
- [RJobLog](#) - rappresenta una lista di messaggi in una registrazione lavori iSeries
- [RMessageQueue](#) - rappresenta una lista di messaggi nella coda messaggi iSeries
- [RPrinterList](#) - rappresenta una lista di stampanti iSeries
- [RUserList](#) - rappresenta una lista di utenti iSeries

Una lista di risorse è sempre o chiusa o aperta. E' necessario che la lista delle risorse sia aperta per accedere al contenuto. Per consentire l'accesso immediato al contenuto della lista e la gestione della memoria in modo efficiente, gran parte delle liste di risorse vengono caricate in modo incrementale.

Le liste delle risorse consentono di:

- [Aprire](#) la lista
- [Chiudere](#) la lista
- [Accedere ad una risorsa specifica](#) dalla lista
- [Attendere il caricamento](#) di una particolare risorsa
- [Attendere il caricamento](#) di una lista di risorse completa

E' anche possibile filtrare le liste delle risorse utilizzando i valori di selezione. Ogni valore di selezione viene identificato utilizzando un ID selezione. Allo stesso modo, è possibile ordinare le liste delle 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 delle liste delle risorse:

- Esempio: [richiamare e stampare il contenuto di una ResourceList](#)
- Esempio: [utilizzare un codice generico per accedere ad una ResourceList](#)
- Esempio: [visualizzare una lista di risorse in un servlet \(tabella HTML\)](#)

# Presentazione classe

Ad ogni oggetto risorsa, lista di risorse e oggetto metadati è associato un oggetto [com.ibm.as400.resource.Presentation](http://com.ibm.as400.resource.Presentation) che fornisce informazioni convertite, come il nome, il nome completo e l'icona.

## Esempio: stampare una lista di risorse e dei relativi valori di ordinamento utilizzando le rispettive Presentation.

E' possibile utilizzare le informazioni Presentation per presentare gli oggetti risorsa, le liste 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 di sicurezza

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 \(Java Secure Socket Extension\)](#) fornisce collegamenti sicuri sia codificando i dati scambiati tra una sessione client e server che eseguendo l'autenticazione del server.

**Nota:** le informazioni relative a [SSL \(Secure Sockets Layer\)](#) sono incluse solo per compatibilità retrograda. 

- [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:** considerare l'utilizzo di [JSSE \(Java Secure Socket Extension\)](#) invece dei metodi che seguono per fornire collegamenti sicuri. Le informazioni che seguono relative all'utilizzo di SSL sono incluse solo per la compatibilità con le versioni precedenti.◀

L'utilizzo di SSL influisce negativamente sulle prestazioni perché i collegamenti SSL sono più lenti dei collegamenti che non dispongono di codifiche. Utilizzare i collegamenti SSL quando per la sensibilità dei dati trasferiti appare opportuno rischiare un rallentamento nelle prestazioni, ad esempio, quando si trasferiscono informazioni su istruzioni di banca o carte di credito.

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. In V5R2, questi algoritmi vengono forniti con il programma su licenza iSeries Client Encryption (128-bit), 5722-CE3.

**Nota:** Toolbox per Java è inoltre compatibile con il programma su licenza iSeries Client Encryption (56 bit), 5722-CE2, il quale non è stato più aggiornato e non è disponibile per V5R2. 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.◀

## Impostare l'ambiente SSL

IBM Toolbox per Java fornisce due ambienti per l'utilizzo di SSL per codificare i dati, che è necessario impostare correttamente.

- [Utilizzare la codifica tra le classi IBM Toolbox per Java e i server OS/400](#)
- [Utilizzare la codifica tra il client proxy e il server proxy](#)

## »Compatibilità con versioni precedenti di IBM Toolbox per Java

La versione V5R2 di IBM Toolbox per Java, gli algoritmi di codifica e i file classe keyring richiedono l'utilizzo sia del programma su licenza V5R1 che V5R2 Client Encryption.

**Nota:** se si esegue l'aggiornamento dalla versione OS/400 V4R5 o precedente, è necessario aggiornare il file KeyRing.class.

Quando si utilizza V5R2 IBM Toolbox per Java e una versione compatibile di Client Encryption sul client, è possibile collegarsi a V4R4 e a versioni più recenti di OS/400. Per ulteriori informazioni relative alle versioni compatibili con 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.



# Utilizzare 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.

## Impostare il client e il 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 per questa fase dipende dal tipo di certificato utilizzato quando si è impostato 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 molto più semplice e veloce di quando si utilizza un'autocertificazione.

3. Utilizzare l'[oggetto SecureAS400](#) per forzare IBM Toolbox per Java a codificare i dati.

**Nota:** il completamento delle due fasi precedenti creano un percorso sicuro tra il client e 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.

# Impostare i 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:** Toolbox per Java è anche compatibile con la versione V5R1 del Cryptographic Access Provider 56-bit per iSeries, 5722-AC2 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 che seguono utilizzati da IBM Toolbox per 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

## Modificare l'autorizzazione dell'indirizzario che contiene i file di codifica 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 agli utenti che dispongono dell'autorizzazione speciale \*ALLOBJ.

## Richiamare e configurare 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.

# Utilizzare 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. E' necessario richiamare i file zip che contengono gli algoritmi di codifica e aggiungerli alla 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.

# Utilizzare 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 IBM Toolbox per Java lo possano utilizzare. « E' necessario richiamare i file zip che contengono gli algoritmi di codifica 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 sia gli algoritmi di codifica che 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 sul client denominato <SSL>\com\ibm\as400\access dove <SSL> è l'indirizzario in cui sono stati copiati i file jar e zip.
5. Dalla richiesta comandi all'interno dell'indirizzario <SSL> sul client, eseguire i comandi:

```
java utilities.KeyringDB com.ibm.as400.access.KeyRing -connect  
<systemname>:<port>
```

dove <port> è la porta server di ognuno dei 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 certificato, è possibile immettere una stringa alfanumerica.

**Nota:** è necessario eseguire KeyringDB su ogni server che dispone di un'autocertificazione per aggiungere ogni certificato alla classe KeyRing. In ogni iSeries in cui si desidera utilizzare i collegamenti SSL, per aggiungere i certificati eseguire il comando:

```
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 nell'istruzione CLASSPATH, è possibile sostituire la classe obsoleta in jt400.jar con la nuova KeyRing.class.

# Utilizzare 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 or 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:** le prime due fasi creano solamente un percorso sicuro tra il client proxy e il 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](#).

# Impostare 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 strumento 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.

## Impostare il 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, il client deve eseguire Java 1.1.8 JVM (e il modulo aggiuntivo Swing 1.1) o Java 2 JVM.

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 in classpath.

## Creare un certificato server

Utilizzare la GUI IKeyman per creare un'autocertificazione.

**Nota:** se la GUI IKeyman arresta l'esecuzione, assicurarsi 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 **Sfoggia** per collocare il file classe che si desidera utilizzare per il keyring.

**Nota:** ricordarsi il nome file del keyring, perché è 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:** ricordarsi la parola d'ordine, necessaria per avviare un 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 sulla lista **Paese** selezionare un paese, 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**.

Ora è possibile visualizzare il keyring creato nell'indirizzario corrente.

## Avviare il 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
```



# Impostare 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 perché il server utilizza un certificato selfsigned.

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](#).

## Impostare il client per utilizzare SSL

Lo strumento che scarica il certificato (KeyringDB) è un programma Java. Per utilizzare questo programma, è necessario eseguire Java 1.1.8 o Java 2 JVM sul client. 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 sul client denominato <SSL>\com\ibm\as400\access dove <SSL> è l'indirizzario in cui sono stati copiati i file jar e zip.

## Aggiungere il 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.

## Impostare il 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 specifica 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.

## Impostare le impostazioni 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 rilascio 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 richiamare [»Certificati Kerberos e token di profilo](#) [«](#) che rappresentano profili utente e parole d'ordine autenticati per il sistema.

**»Nota:** l'utilizzo di certificati Kerberos richiede l'installazione di J2SDK, v1.4 e la configurazione dell'API JGSS (Java General Security Services). Per ulteriori informazioni su JGSS, consultare [J2SDK, v1.4 Security Documentation](#)



**»** Per utilizzare i certificati Kerberos, impostare solo il nome sistema (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 annulla ogni certificato Kerberos o token di 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.

**»** L'esempio che segue crea un oggetto di sistema e lo utilizza 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); «
```

## Impostare le 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 è più sicuro rispetto al passare il profilo utente e la parola d'ordine grazie alla durata limitata, i token di

profilo devono essere ancora considerati come informazioni sensibili 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.

## Esempio

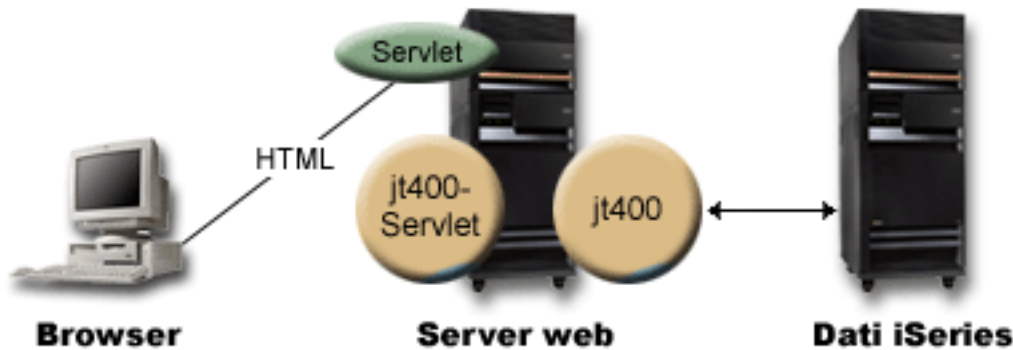
Fare riferimento a questo [codice](#) per un esempio di come utilizzare la credenziale del token di profilo per scambiare l'identità del sottoprocesso OS/400 ed eseguire le funzioni per conto di un utente specifico.

# 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 nel server web dal momento che le classi servlet utilizzano alcune classi access per richiamare i dati e classi HTML per presentare i dati. Il server web è collegato al sistema iSeries in cui si trovano i dati.

**Figura 1: come funzionano i servlet**



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 desidera 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](#).

# 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. Le sottoclassi di AuthenticationServlet dovrebbero sostituire uno o più dei metodi che seguono:

- 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 le [eccezioni](#) e i [messaggi](#)

## Classi AS400Servlet

La classe [AS400Servlet](#) è una sottoclasse astratta di AuthenticationServlet che rappresenta un servlet HTML. È possibile utilizzare un [lotto di collegamenti](#) per condividere i collegamenti e gestire il numero di collegamenti di cui un utente servlet può disporre sul server.

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 [riferimento](#).

# Classe RowData

La classe [RowData](#) è una classe astratta che fornisce un metodo per descrivere e accedere ad una lista 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 nella lista utilizzando il metodo [length\(\)](#).

## Posizione RowData

Esistono diversi metodi che consentono di richiamare e impostare la posizione corrente in una lista. La tabella che segue elenca i metodi di impostazione e di richiamo per le classi RowData.

Metodi Set		Metodi Get
<a href="#">absolute()</a>	<a href="#">next()</a>	<a href="#">getCurrentPosition()</a>
<a href="#">afterLast()</a>	<a href="#">previous()</a>	<a href="#">isAfterLast()</a>
<a href="#">beforeFirst()</a>	<a href="#">relative()</a>	<a href="#">isBeforeFirst()</a>
<a href="#">first()</a>		<a href="#">isFirst()</a>
<a href="#">last()</a>		<a href="#">isLast()</a>

# Classe ListRowData

La classe ListRowData consente di effettuare le seguenti operazioni:

- [Aggiungere](#) e [rimuovere](#) le righe a/da una lista di risultati.
- [Richiamare](#) e [impostare](#) la riga
- Richiamare informazioni sulle colonne della lista con il metodo [getMetaData\(\)](#)
- Impostare le informazioni relative alla colonna con il metodo [setMetaData\(\)](#)

La classe [ListRowData](#) rappresenta una lista di dati. ListRowData può rappresentare molti tipi di informazioni, incluse le seguenti, tramite le [classi Access](#) di IBM Toolbox per Java:

- Un indirizzario nell'[IFS \(Integrated File System\)](#)
- Una lista di [lavori](#)
- Una lista di messaggi in una [coda messaggi](#)
- Una lista di [utenti](#)
- Una lista di [stampanti](#)
- Una lista di [file di spool](#)

Questo [esempio](#) mostra come funzionano ListRowData e HTMLTableConverter. Mostra il codice java, il codice HTML e l'aspetto dell'HTML.



# Classe RecordListRowData

La classe RecordListRowData consente di effettuare le seguenti operazioni:

- [Aggiungere](#) ed [eliminare](#) righe alla/dalla la lista di record.
- [Richiamare](#) ed [impostare](#) la riga
- Impostare il formato record con il metodo [setRecordFormat](#).
- Richiamare il formato record .

La classe [RecordListRowData](#) rappresenta una lista di 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 una lista di risorse di dati. Utilizzare gli oggetti ResourceListRowData per rappresentare qualsiasi implementazione dell'interfaccia [ResourceList](#).

Le liste di risorse vengono formattate 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](#) la lista di risorse
- [Richiamare il numero di righe](#) nella lista
- [Richiamare i dati colonna](#) per la riga corrente
- [Richiamare la lista proprietà](#) dell'oggetto dati
- [Richiamare i metadati](#) per la lista

**Esempio:** [presentare una lista di risorse in un servlet](#)❏

# Classe `SQLResultSetRowData`

La classe [SQLResultSetRowData](#) rappresenta un insieme di risultati SQL come una lista di dati. Questi dati vengono creati da una istruzione SQL attraverso [JDBC](#). Assieme ai metodi forniti, è possibile [richiamare](#) 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 siano [dati 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

[ListMetaData](#) consente di ottenere informazioni sulle colonne e di modificarne le impostazioni in una classe [ListRowData](#). Essa utilizza il metodo [setColumns\(\)](#) per impostare il numero di colonne, eliminando il contenuto di tutte le precedenti informazioni sulla colonna. Alternativamente, è possibile anche immettere il numero di colonne quando si impostano i parametri del programma di creazione.

Questo [esempio](#) mostra come funzionano ListMetaData, ListRowData e HTMLTableConverter. Mostra il codice java, il codice HTML e l'aspetto dell'HTML.

# Classe RecordFormatMetaData

La classe [RecordFormatMetaData](#) utilizza la classe [RecordFormat](#) di 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 convertitore

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](#)

## Esempi

Esempio: [utilizzare HTMLFormConverter](#). (Compilare ed eseguire questo esempio con un server web attivo)

# Classe HTMLTableConverter

La classe [HTMLTableConverter](#) estende [StringConverter](#) fornendo un metodo [convertToTables\(\)](#). Questo metodo converte i dati di riga in una schiera di tabelle HTML che il servlet può utilizzare per visualizzare la lista 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: [utilizzare ListRowData](#)
- Esempio: [utilizzare RecordListRowData](#)
- Esempio: [utilizzare ResultSetRowData](#)
- Esempio: [presentare ResourceList in un servlet](#)

# Classi di utilità

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.
- [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 le 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.

## Utilizzare 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 Toolbox sul client che per mantenerli aggiornati. Quando vengono utilizzati per la prima volta, i file Toolbox vengono copiati dal server sulla stazione di lavoro. Quando si applicano le PTF sul 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.

## Eseguire la 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 Toolbox o solo alcune funzioni. » Ad esempio, per installare o aggiornare classi nel pacchetto Access Toolbox per Java (jt400.jar) sulla propria stazione di lavoro, utilizzare i comandi che seguono:

```
java utilities.AS400ToolboxInstaller -install -package ACCESS -source  
myAS400 -target c:\toolbox
```

L'esempio precedente presuppone che c:\toolbox sia l'indirizzario che contiene i file jar Toolbox per Java. «

## 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

## Utilizzare AS400ToolboxJarMaker

### Decomprimere 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
```

### Dividere un file JAR singolo in molti file JAR più piccoli

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 300K:

```
java utilities.AS400ToolboxJarMaker -split 300
```

**Rimuovere file non utilizzati dal file JAR** Con [AS400ToolboxJarMaker](#), è possibile escludere qualsiasi file IBM Toolbox per Java che non sia necessario all'applicazione, selezionando solo i [componenti](#) IBM Toolbox per Java, il linguaggio e [CCSID](#) necessari per eseguire la propria 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 400K 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 Program Call. Quando si includono le classi Program Call, è necessario includere esplicitamente le classi di conversione utilizzate dal programma, utilizzando l'opzione -ccsid.



# Componenti supportati da IBM Toolbox per Java

Di seguito sono riportati gli ID dei componenti che possono essere specificati quando si richiama lo strumento AS400ToolboxJarMaker.

- La prima colonna elenca il nome comune di ogni singolo componente.
- La seconda colonna elenca le parole chiave che è necessario specificare quando si utilizza la tag di opzione -component.
- La terza colonna 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	Trace	AS400ToolboxJarMaker.TRACE
Utenti e gruppi	User	AS400ToolboxJarMaker.USER
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

IBM Toolbox per Java viene fornito con una serie di tabelle di conversione, denominate in base al CCSID. Queste tabelle vengono utilizzate internamente dalle classi IBM Toolbox per Java (come CharConverter) quando si convertono dati che vengono spostati da o su un server iSeries o AS/400e. 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 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 iSeries e AS/400e.

Per ulteriori informazioni relative ai CCSID, inclusa una lista completa di CCSID riconosciuti dai server iSeries e AS/400e, consultare l'[argomento Globalizzazione](#) nell'Information Center.

## 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
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
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 (sopperto 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)
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 e AS/400e (utilizzato principalmente in IFS)
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 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
- x4j400.jar
- jhall.jar

Tutti i file jar, ad eccezione del file jhall.jar, sono inclusi in Toolbox per Java. Per ulteriori informazioni sui file jar di Toolbox per Java, consultare [File jar](#). Per ulteriori informazioni su come scaricare 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.

**Esempio:** [Utilizzare CommandPrompter con le classi CommandCall e AS400Message per richiedere ed eseguire un comando](#) 

# 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 emmissione per emmissione 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: utilizzare 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 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 Swing 1.1. E' possibile ottenere Swing 1.1 sia eseguendo Java 2 che scaricando Swing 1.1 da [Sun Microsystems, Inc.](#) . In passato, IBM Toolbox per Java richiedeva Swing 1.0.3 e V4R5 è il primo rilascio in cui Swing 1.1 viene supportato. Per spostarsi su Swing 1.1, sono state effettuate alcune modifiche di programmazione; tuttavia è possibile che anche l'utente stesso debba fare delle modifiche di programmazione. Per ulteriori informazioni relative a Swing consultare la pagina di [Sun Microsystems, Inc. JFC](#) .

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\\*](#)
- [IFS](#)
- [JavaApplicationCall](#)
- [JDBC](#)
- [Lavori\\*](#)
- [Messaggi\\*](#)
- [Autorizzazione](#)
- [Stampa\\*](#) incluso [il programma di visualizzazione file di spool](#)
- [ProgramCall e ProgramParameter](#)
- [Accessi al livello record](#)
- [Liste 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.

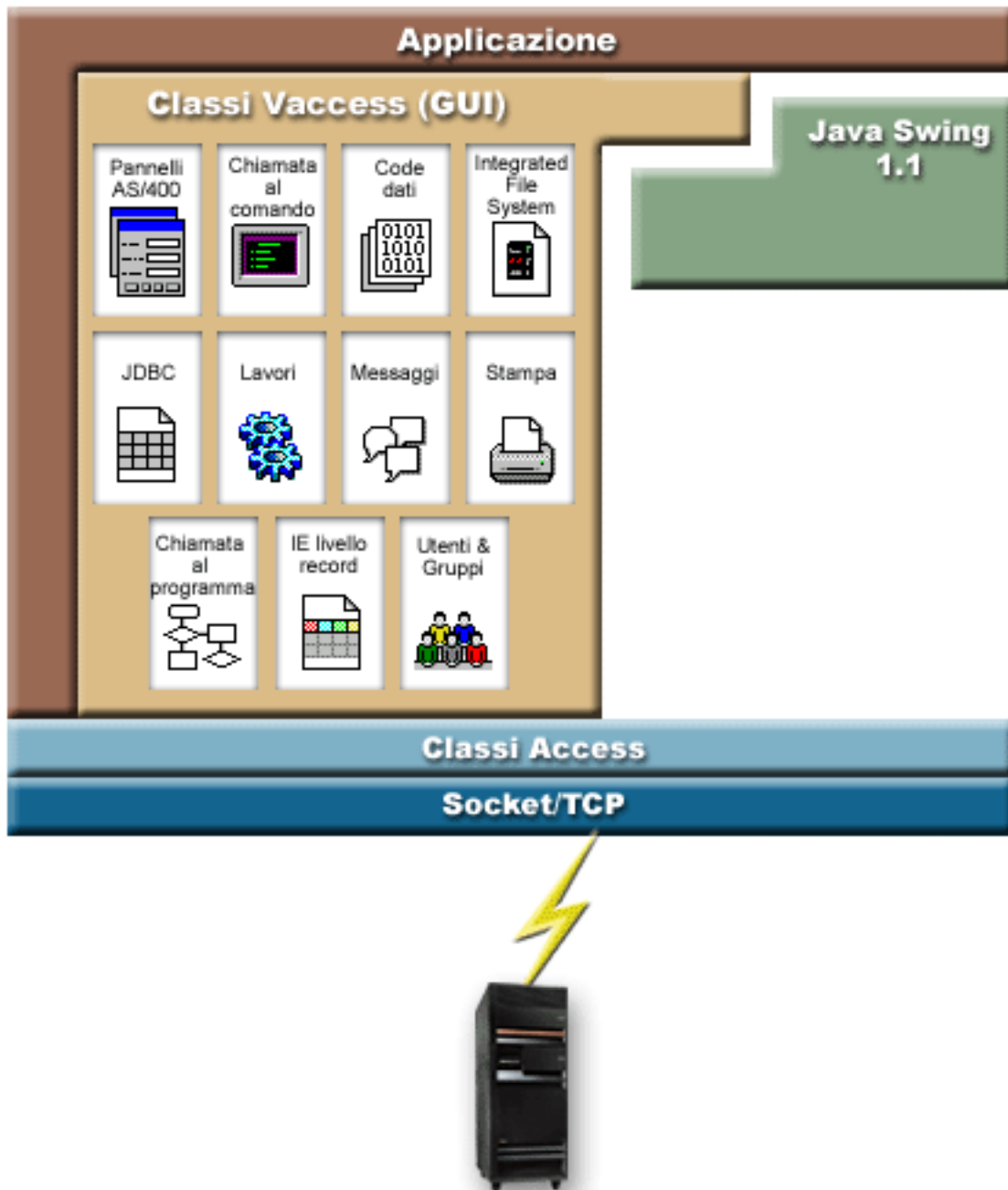
Ogni volta che si esegue una programmazione con i componenti GUI di IBM Toolbox per Java, utilizzare le classi di Eventi errore per notificare all'utente gli eventi errore e per gestirli.

Per ulteriori informazioni relative all'accesso ai dati iSeries, consultare [Classi Access](#).

# 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**



# 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 una lista 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 una lista 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 del 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, rispettivamente, un lavoro o una lista di lavori
- [VMessageList](#) e [VMessageQueue](#) rappresentano, rispettivamente, una lista di messaggi restituiti da un CommandCall, da un ProgramCall o da una coda messaggio
- [VPrinter](#), [VPrinters](#) e [VPrinterOutput](#) rappresentano, rispettivamente, una stampante, una lista di stampanti o una lista di file di spool
- [VUserList](#) rappresenta una lista di utenti

Tutte le risorse sono implementazioni dell'interfaccia [VNode](#).

## Impostare la 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 nella propria lista.
- [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 la lista di utenti definiti sul sistema:

```
// Creare la risorsa server
// che rappresenta una lista di utenti.
// Supponiamo che "system" sia un oggetto
// 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
// la lista utenti.
AS400DetailsPane detailsPane = new AS400DetailsPane ();
detailsPane.setRoot (userList);

// Aggiungere il pannello dettagli ad una frame.
```

```
        // Presupporre che "frame" sia un JFrame
// creato da qualche altra parte.
frame.getContentPane ().add (detailsPane);
```

## Caricare il 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 un segmento:

```
        // Caricare il contenuto del pannello
        // dettagli. Supponiamo che il detailsPane
        // sia stato creato ed inizializzato da qualche altra
// da qualche altra parte.
        detailsPane.load ();

// Aggiungere il pannello dettagli ad una frame.
        // Presupporre 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 una lista 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.

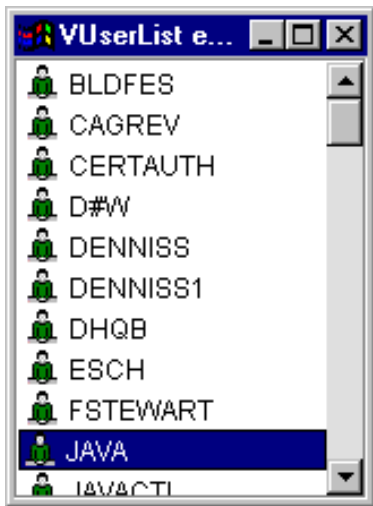
I seguenti modelli sono disponibili:

- [AS400ListModel](#) implementa l'interfaccia ListModel JFC come lista 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

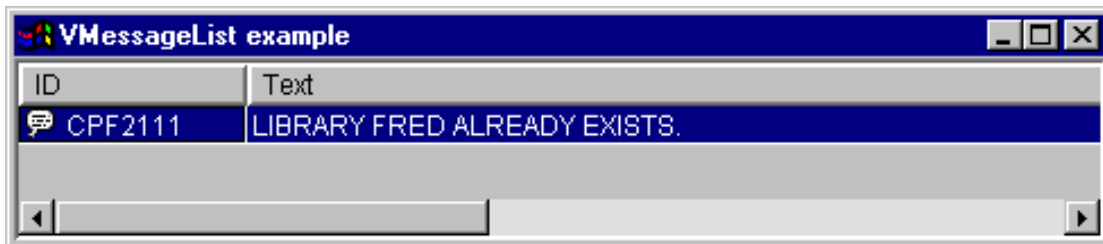
- Presenta una lista di utenti sul sistema utilizzando un [AS400ListPane](#) con un oggetto VUserList. La figura 1 illustra il prodotto finito:

**Figura 1: utilizzo di AS400ListPane con un oggetto VUserList**



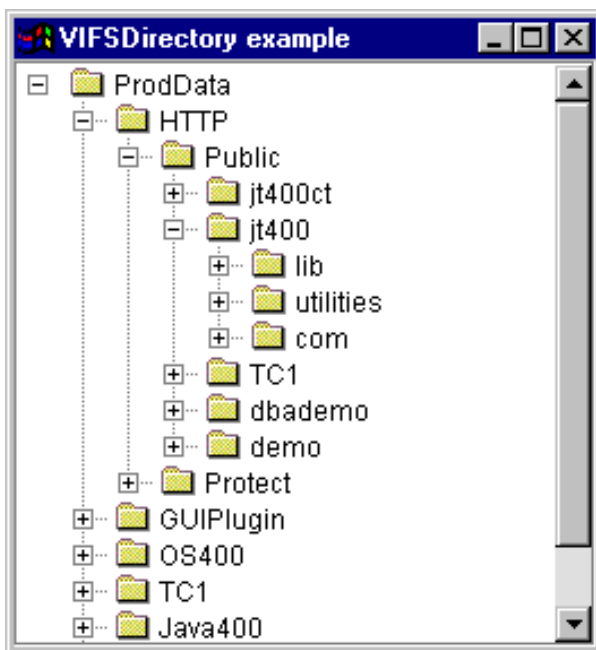
- Presentare la lista dei messaggi creati da una chiamata al comando utilizzando un [AS400DetailsPane](#) con un oggetto VMessageList. La figura 2 illustra il prodotto finito:

**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**

The image shows a window titled "VPrinters example" with a standard Windows-style title bar (minimize, maximize, close buttons). On the left, there is a tree view under "Printers" containing three items: "JAVABLDA", "JAVABLDB", and "OS2VPRT". The main area of the window displays a table with three columns: "Printer", "Status", and "Description".

Printer	Status	Description
JAVABLDA	Powered off or not yet available	DEVICE CREATED FC
JAVABLDB	Powered off or not yet available	DEVICE CREATED FC
OS2VPRT	Stopped	DEVICE CREATED FC

# 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 `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 ai comandi, impostare le proprietà sistema e comando. Tali proprietà possono essere impostate utilizzando un programma di creazione o tramite i metodi `setSystem()` e `setCommand()`.

L'esempio seguente crea un `CommandCallButton`. Durante l'esecuzione, quando si preme il pulsante, viene creata una libreria denominata "FRED":

```
// Creare l'oggetto CommandCallButton.
// Presupporre che "system" sia un oggetto
// AS400 creato e inizializzato da qualche
// altra parte. Il testo del 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 ("CTRLIB FRED");

// Aggiungere il pulsante ad una frame. Presupporre
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 generati dal comando:

```
// 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 la lista dei messaggi server
        // generati dal comando.
        AS400Message[] messageList = sourceButton.getMessageList ();

        // ... Elaborare la lista dei messaggi.

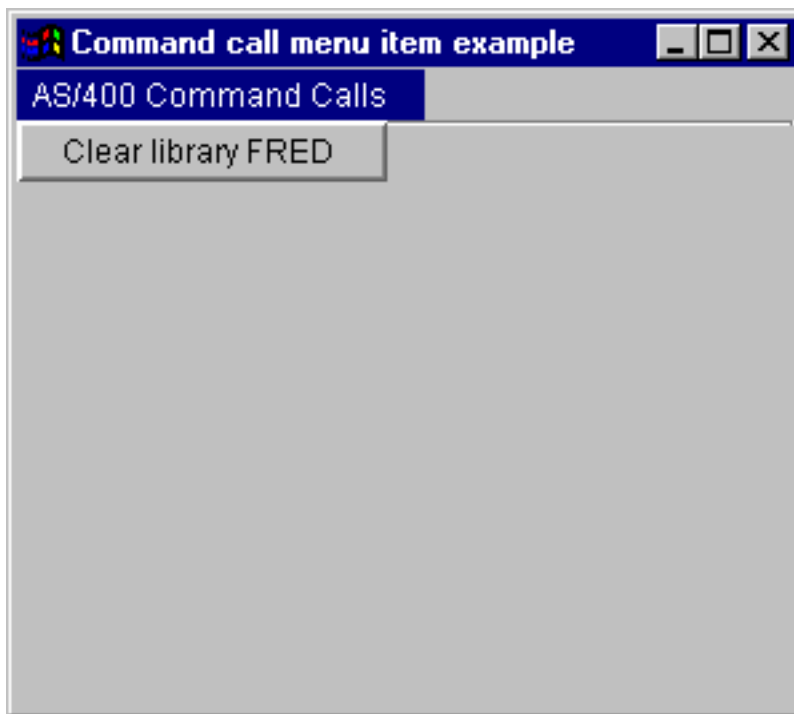
    });
}
```

## 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 di 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** e [KeyedDataQueueDocument](#) 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.

L'esempio seguente crea un [DataQueueDocument](#) il cui contenuto viene associato alla coda dati:

```
// Creare l'oggetto DataQueueDocument.
// Presupporre 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.
// Presupporre che "frame" sia un JFrame
// creato da qualche altra parte.
frame.getContentPane ().add (textArea);
```

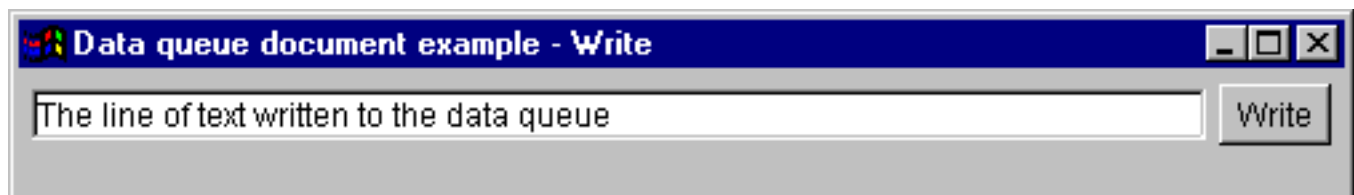
Inizialmente, il contenuto del componente testo è vuoto. Utilizzare `read()` o `peek()` per riempire il contenuto con la voce successiva nella coda. Utilizzare `write()` per scrivere il contenuto del componente testo nella coda dati. Tenere presente che tali documenti funzionano solamente con voci coda dati `String`.

## 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: gestire gli eventi errore tramite la visualizzazione di una finestra di dialogo

Il seguente esempio mostra come gestire gli eventi errore visualizzando una finestra di dialogo:

```
        // ... tutto il lavoro per il layout
        // di un componente graphical user
        // interface è stato svolto. Ora aggiungere un
        // ErrorDialogAdapter come listener
        // al componente. Questo notificherà
        // tutti gli eventi errore lanciati da quel
        // componente tramite la visualizzazione di una
        // finestra di dialogo.
        ErrorDialogAdapter errorHandler = new ErrorDialogAdapter (parentFrame);
        component.addErrorListener (errorHandler);
```

E' possibile scrivere un listener errore di personalizzazione per gestire gli errori in un modo differente. Utilizzare l'interfaccia [ErrorListener](#) per effettuarlo.

### Esempio: definire un listener di errore

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 volta
    // che viene lanciato un evento di errore.
    public void errorOccurred(ErrorEvent event)
    {
        Exception e = event.getException ();
        System.out.println ("Error: " + e.getMessage ());
    }
}
```

### Esempio: gestire gli 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.

Sono disponibili i seguenti componenti:

- [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`).

E' necessario impostare il percorso di un indirizzario su un elemento diverso da `"/QSYS.LIB"` poiché tale indirizzario è solitamente di grosse dimensioni e lo scaricamento del suo contenuto potrebbe richiedere diverso tempo.

# 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
// oggetto AS/400 e che "frame" sia un JFrame
// ed inizializzata altrove.
IFSFileDialog dialog = new IFSFileDialog (frame, "Select a file",
system);

// Impostare una lista 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");

// Mostrare la finestra di dialogo. Se l'utente ha
// selezionato un file premendo il pulsante
// "Apri", 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**

**File Open** ✕

Dircoctry

.  
..  
com  
lib  
utilities

File

ACCESS.LST  
ACCESS.LVL  
JT400.PKG  
V3R2M1.LST

Open

Cancel

//rchas1 dd/QIBM/ProdData/HTTP/Public/jt400

File name:

File type:  ▼

Ready

# Indirizzari in AS400Panee

[AS400Panee](#) sono componenti GUI che presentano e consentono la gestione di una o più risorse server. Un oggetto [VIFSDirectory](#) è una risorsa che rappresenta un indirizzario nell'IFS (Integrated File System) da utilizzare in AS400Panee. 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 AS400Panee. 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.
        // Presupporre che "system" si trovi in un
// 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 ();

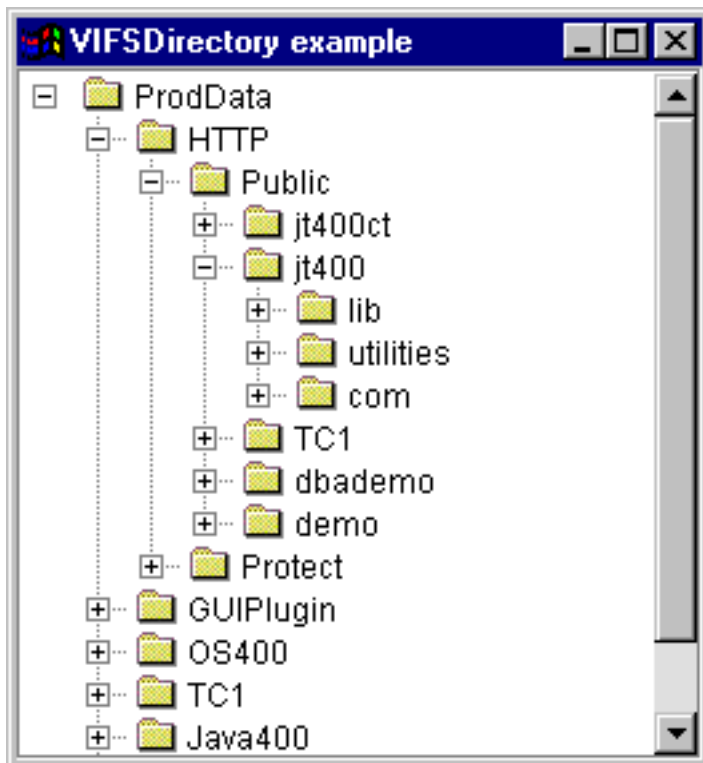
        // Aggiungere il pannello explorer ad una frame.
        // Presupporre che "frame" sia un JFrame
// creato da qualche altra parte.
frame.getContentPane ().add (explorerPane);
```

## Esempio

Visualizza 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 documento di testo, solitamente utilizzando il programma di creazione del componente testo o con 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
        // IFSTextFileDocument. Presupporre che
        // "system" sia un oggetto AS400
        // 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.
        // Presupporre 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 documento del file di testo IFS**

IFS text file document example



File

Welcome to the AS/400 Toolbox for Java. This file shows the capabilities of the IFSTextFileDocument class. Give it a try!

# 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).

Sono disponibili i seguenti componenti:

- [SQLStatementButton](#) e [SQLStatementMenuItem](#) sono, rispettivamente, un pulsante o una voce di menu che emette un'istruzione SQL quando selezionata.
- [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. L'esempio seguente registra l'unità di controllo JDBC di AS/400 Toolbox per Java:

```
// Registrare l'unità di controllo JDBC.  
DriverManager.registerDriver(new  
com.ibm.as400.access.AS400JDBCdriver());
```

## Collegamenti SQL

Un oggetto [SQLConnection](#) 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 [setProperty\(\)](#) 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. Presupporre
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 di testo, utilizzando di solito il programma di creazione del componente di 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.
// Supponiamo che "connection" sia
// sia un oggetto SQLConnection creato
// ed inizializzata 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 l'impostazione risultato sia chiusa.

Il seguente esempio crea un oggetto `SQLResultSetFormPane` e lo aggiunge ad un segmento:

```
        // Creare un oggetto SQLResultSetFormPane.
        // Supponiamo che "connection" sia
        // sia un oggetto SQLConnection creato
        // ed inizializzata 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.
// Supponiamo che "connection"
// sia un oggetto SQLConnection creato
// ed inizializzata altrove.
SQLResultSetTablePane tablePane = new SQLResultSetTablePane
(connection, "SELECT * FROM QIWS.QCUSTCDT");

// Caricare i risultati.
tablePane.load ();

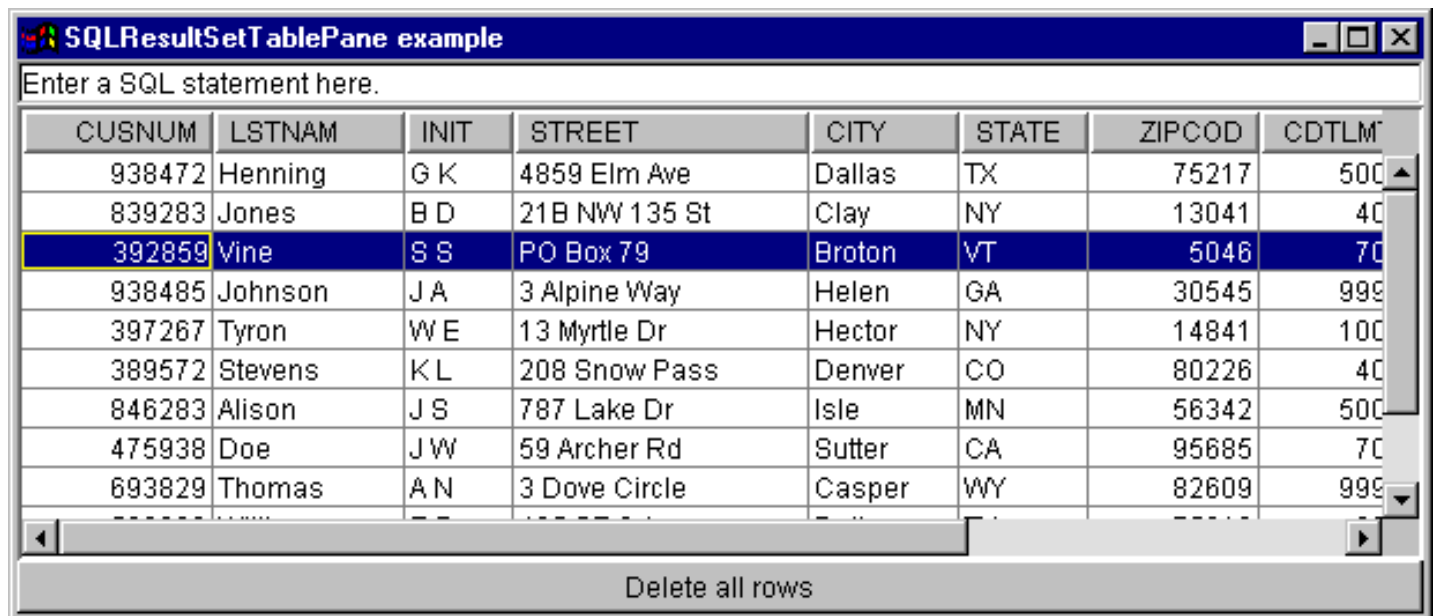
// Aggiungere il pannello tabella ad una frame.
// Presupporre che "frame" sia un JFrame
// creato da qualche altra parte.
frame.getContentPane ().add (tablePane);
```

## 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.
        // Supponiamo che "connection" sia
        // sia un oggetto SQLConnection creato
        // ed inizializzata 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.
        // Supponiamo che "connection" sia
        // sia un oggetto SQLConnection creato
        // ed inizializzata 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
dell'interrogazione
        // ad una frame. Supponiamo che "frame" sia un
        // 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](#)**

SQLQueryBuilderPane example

Tables | Select | Join By | Where | Group By | Having | Order By | Summary

Catalog:

Set schemas

Schema	Table	Type	Description
QIWS	QAZDCOLM	TABLE	CATALOG - SYSCOLUMNS, C
QIWS	QAZDGCOL	TABLE	CATALOG - SYSCOLUMNS, C
QIWS	QAZDGTB1	TABLE	CATALOG - SYSTABLES, ALL
QIWS	QAZDGTB4	TABLE	CATALOG - SYSTABLES, ALL
QIWS	QAZDGTB5	TABLE	CATALOG - SYSTABLES, ALL
QIWS	QAZDGTB7	TABLE	CATALOG - SYSTABLES, ALL
QIWS	QAZDTBL1	TABLE	CATALOG - SYSTABLES, ALL
QIWS	QAZDTBL2	TABLE	CATALOG - SYSTABLES, PH
QIWS	QAZDTBL3	TABLE	CATALOG - SYSTABLES, PH
QIWS	QAZDTBL4	TABLE	CATALOG - SYSTABLES, PH
QIWS	QAZDTBL5	TABLE	CATALOG - SYSTABLES, PH

Tables

QIWS.QCUSTCDT

Show result set

# Lavori

I componenti vaccess (GUI) dei lavori consentono ad un programma Java di presentare le liste dei lavori del server e dei messaggi di registrazione lavori in una GUI.

Sono disponibili i seguenti componenti:

- Un oggetto [VJobList](#) è una risorsa che rappresenta una lista di lavori del server da utilizzare in [AS400Panels](#).
- Un oggetto [VJob](#) è una risorsa che rappresenta la lista dei messaggi nella registrazione lavori da utilizzare in AS400Panels.

E' possibile utilizzare insieme gli oggetti AS400Panels, VJobList e VJob per presentare molte viste di una lista 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 oppure tramite 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 AS400Panels. Utilizzare [setName\(\)](#) per specificare che è necessario visualizzare solo lavori con un determinato nome. Utilizzare [setNumber\(\)](#) per specificare che è necessario visualizzare solo lavori con un determinato numero. Utilizzare allo stesso modo [setUser\(\)](#) per specificare che è necessario visualizzare solo i lavori di un determinato utente.

Quando vengono creati, gli oggetti AS400Pane, VJobList e VJob, sono inizializzati su uno stato predefinito. La lista 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 della lista.

Al momento dell'esecuzione, fare clic con il tastino destro del mouse su un lavoro, una lista lavori o un messaggio di registrazione lavori per visualizzare il menu di scelta rapida. Selezionare **Proprietà** da menu di scelta rapida per eseguire delle operazioni sull'oggetto selezionato:

- Lavoro - Gestire la proprietà, come il tipo e lo stato. E' inoltre possibile modificare il valore di alcune delle proprietà.
- Lista lavori - Gestire la proprietà, come il nome, numero e proprietà utente. E' inoltre possibile modificare il contenuto della lista.
- 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 azioni utilizzando il metodo [setAllowActions\(\)](#) sul pannello.

Il seguente esempio crea un VJobList e lo presenta in un AS400ExplorerPane:

```
// Creare l'oggetto VJobList. Supponiamo
// che "system" sia un oggetto AS400 creato
// ed inizializzata 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 una lista di lavori. La lista visualizza i lavori sul sistema che hanno lo stesso nome lavoro.

La seguente immagine visualizza il componente GUI VJobList:

Job list example					
Job name	User	Job nu...	Subsystem	Type	
QZDASOINIT	QUSER	014994	/QSYS/QSERVER	Batch	
QZDASOINIT	QUSER	015026	/QSYS/QSERVER	Batch	
QZDASOINIT	QUSER	016098	/QSYS/QSERVER	Batch	
QZDASOINIT	QUSER	016132	/QSYS/QSERVER	Batch	
QZDASOINIT	QUSER	016133	/QSYS/QSERVER	Batch	
QZDASOINIT	QUSER	016144	/QSYS/QSERVER	Batch	
QZDASOINIT	QUSER	016234	/QSYS/QSERVER	Batch	
QZDASOINIT	QUSER	016299	/QSYS/QSERVER	Batch	
QZDASOINIT	QUSER	016300	/QSYS/QSERVER	Batch	
QZDASOINIT	QUSER	016301	/QSYS/QSERVER	Batch	
QZDASOINIT	QUSER	016302	/QSYS/QSERVER	Batch	
QZDASOINIT	QUSER	016303	/QSYS/QSERVER	Batch	
QZDASOINIT	QUSER	016304	/QSYS/QSERVER	Batch	
QZDASOINIT	QUSER	016305	/QSYS/QSERVER	Batch	
QZDASOINIT	QUSER	016314	/QSYS/QSERVER	Batch	

# Classi di messaggio Vaccess

I componenti GUI dei messaggi consentono a un programma Java di presentare le liste dei messaggi del server nella GUI.

Sono disponibili i seguenti componenti:

- Un oggetto [Lista messaggi](#) è una risorsa che rappresenta una lista di messaggi da utilizzare in AS400Panels. Ciò viene creato per le liste 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 AS400Panels.

[AS400Panels](#) 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 le liste dei messaggi del server in AS400Panels.

E' possibile utilizzare gli oggetti `AS400Panel`, `VMessageList` e `VMessageQueue` insieme per presentare molte viste di una lista 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 una lista di messaggi da utilizzare in [AS400Panes](#). Ciò viene creato per le liste dei messaggi da un comando o dalle chiamate al programma. Il seguente metodo restituisce le liste 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. La lista 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
        // da qualche altra parte.
        VMessageList root = new VMessageList (command.getMessageList ());

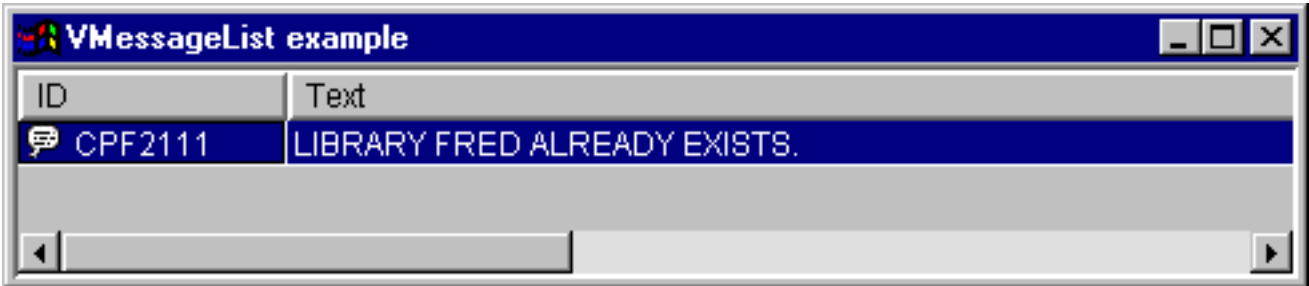
        // Creare e caricare un oggetto
        // AS400TreePane.
AS400DetailsPane detailsPane = new AS400DetailsPane (root);
        detailsPane.load ();

        // Aggiungere il pannello dettagli ad una frame.
        // Presupporre che "frame" sia un JFrame
        // creato da qualche altra parte.
        frame.getContentPane ().add (detailsPane);
```

## Esempio

Presentare la lista 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**



The image shows a window titled "VMessageList example" with a standard Windows-style title bar (minimize, maximize, close buttons). The window contains a table with two columns: "ID" and "Text". The first row of the table is highlighted in blue and contains the values "CPF2111" and "LIBRARY FRED ALREADY EXISTS." respectively. Below the table is a horizontal scrollbar.

ID	Text
CPF2111	LIBRARY FRED ALREADY EXISTS.

# 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 che dovranno essere visualizzati. Utilizzare [setSelection\(\)](#) per specificare il tipo di messaggi che dovranno essere visualizzati.

Quando vengono creati gli oggetti AS400Pane e VMessageQueue, sono inizializzati su uno stato predefinito. La lista 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 della lista.

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 della lista

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
// 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.
                // Presupporre che "frame" sia un JFrame
// creato da qualche altra parte.
frame.getContentPane ().add (explorerPane);
```

## Esempio

Presentare la lista 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**

Message queue example					
JAVACTL	ID	Text	Severity	Type	Date
	CPF1241	JOB 016029/QUSER/QGYSE...	0	Completion	18-Mar-98 3:
	CPF1241	JOB 015924/QUSER/QGYSE...	0	Completion	18-Mar-98 2:
	CPF3390	WRITER 014744/QSPLJOB/...	0	Informational	16-Mar-98 8:
	CPF3453	WRITER OS2VPRT FINISHE...	60	Informational	16-Mar-98 8:
	CPF3382	WRITER 014744/QSPLJOB/...	0	Informational	16-Mar-98 8:
	CPF1241	JOB 014038/QUSER/QGYSE...	0	Completion	12-Mar-98 2:
	CPF1241	JOB 013720/QUSER/QGYSE...	0	Completion	11-Mar-98 6:
	CPF1241	JOB 013295/JAVACTL/QJVA...	0	Completion	11-Mar-98 9:

# 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 liste delle risorse di stampa del server nella GUI:

- Un oggetto [VPrinters](#) è una risorsa che rappresenta una lista 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 una lista 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 le liste 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 una lista 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. La lista 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 della lista 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 della lista.

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
// 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.
// Presupporre 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**

**VPrinters example**

Printers

- JAVABLDA
- JAVABLDB
- OS2VPRT

Printer	Status	Description
JAVABLDA	Powered off or not yet available	DEVICE CREATED FC
JAVABLDB	Powered off or not yet available	DEVICE CREATED FC
OS2VPRT	Stopped	DEVICE CREATED FC



## 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 la lista 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 della lista.

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
// 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.
                // Presupporre 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**

**VPrinter Example**

OS2VPRT

Output name	User-specified data	User	Status	Print
QPJOBLOG	QPADEV0001	JAVACTL	Message waiting	OS2\
QPJOBLOG	QPADEV0001	JA	Ready	OS2\
QPJOBLOG	QPADEV0004	JA	Ready	OS2\
QPJOBLOG	QPADEV0004	JA	Ready	OS2\
QPJOBLOG	QPADEV0001	JA	Ready	OS2\
QPJOBLOG	QPADEV0001	JA	Ready	OS2\
QPJOBLOG	QPADEV0001	JA	Ready	OS2\

- Reply
- Hold**
- Release
- Print next
- Send
- Move
- Delete
- Properties

# Classe VPrinterOutput

Un oggetto [VPrinterOutput](#) è una risorsa che rappresenta una lista di file di spool sul server da utilizzare in [AS400Panels](#).

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 AS400Panels. 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. La lista dei file di spool non viene caricata 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 della lista.

Al momento dell'esecuzione, un utente può eseguire le operazioni su ogni file di spool o sulla lista del file di spool facendovi clic con il tastino destro del mouse per visualizzare il menu di contesto. E' possibile che il menu di contesto della lista di file di spool comprenda una voce denominata **Proprietà** che consente all'utente di impostare le proprietà di filtro che possono modificare il contenuto della lista.

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
// creato ed inizializzato
// da qualche altra parte.
VPrinterOutput root = new VPrinterOutput (system);

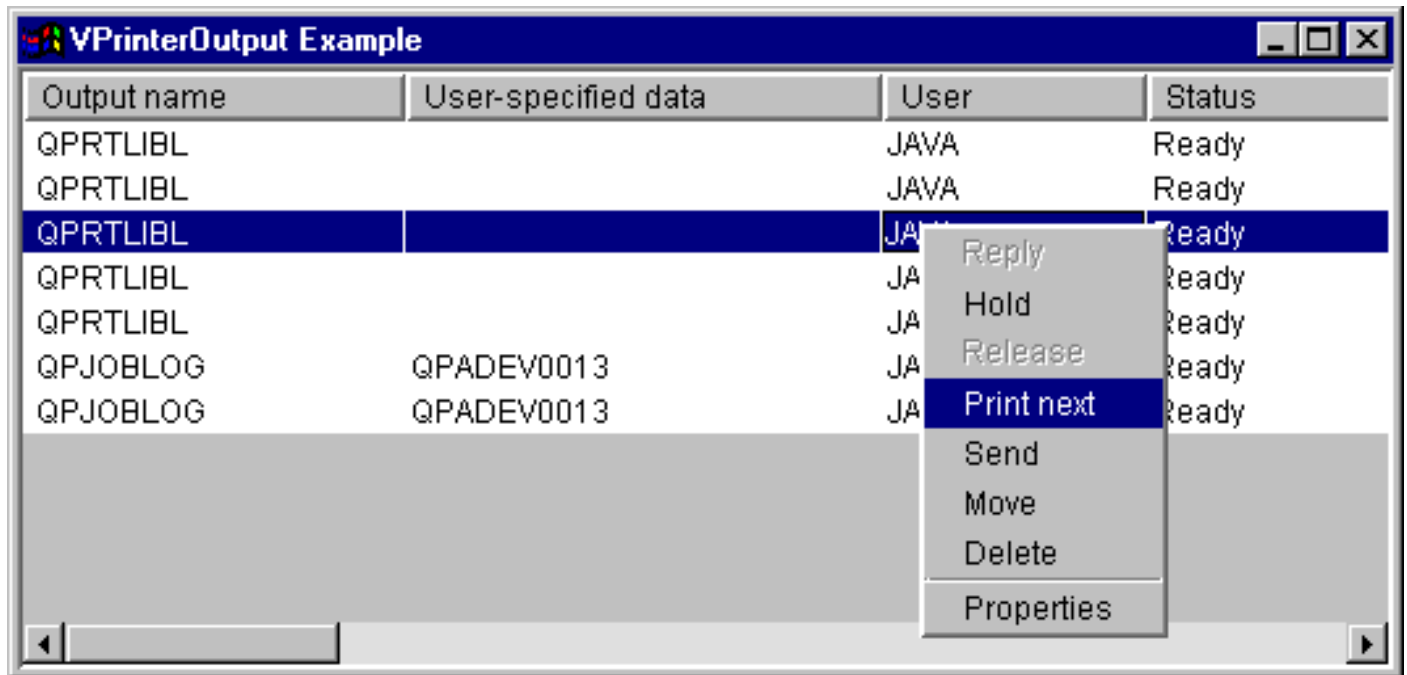
        // Creare e caricare un oggetto
        // AS400TreePane.
AS400ListPane listPane = new AS400ListPane (root);
        listPane.load ();

        // Aggiungere il pannello lista ad una frame.
        // Presupporre che "frame" sia un JFrame
// creato da qualche altra parte.
frame.getContentPane ().add (listPane);
```

## Esempio

Presentare una lista 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).

## Utilizzare la 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 concluderà con una chiamata al metodo [close\(\)](#) che chiude il flusso di immissione e rilascia qualsiasi associazione di risorsa al flusso.

## Utilizzare SpooledFileViewer

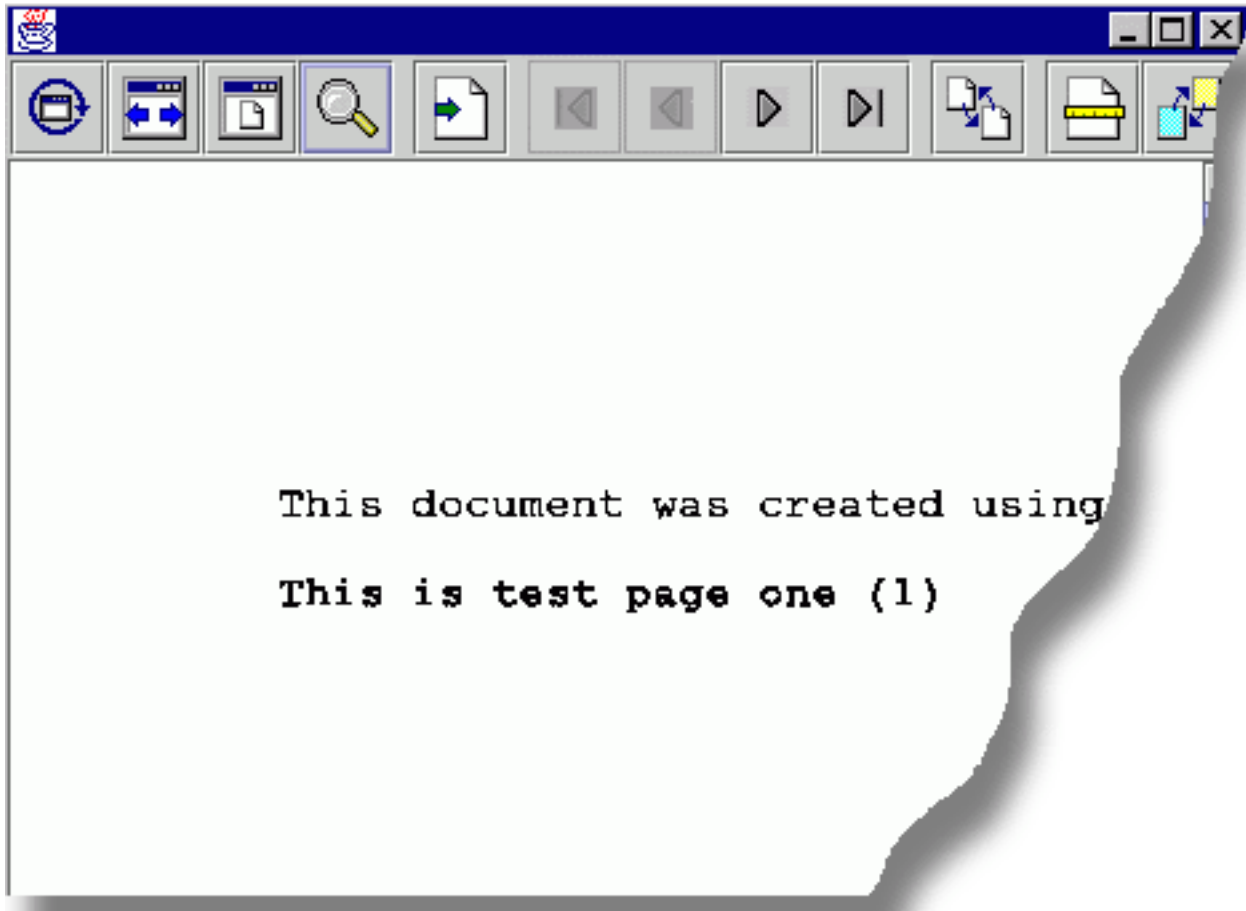
Un'istanza della classe SpooledFileViewer è effettivamente una rappresentazione grafica di un programma di visualizzazione capace di visualizzare ed esplorare un file di spool AFP o SCS. Ad esempio, il codice che segue crea il programma di visualizzazione file di spool nella Figura 1 per visualizzare un file di spool creato in precedenza sul server.

**Nota:** per una spiegazione delle sue funzioni, è possibile selezionare sia il pulsante sull'immagine della [Figura 1](#), che

(se il browser non è abilitato su 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 della barra degli strumenti di SpooledFileViewer



Il pulsante Dimensione reale riporta l'immagine della pagina del file di spool alla sua dimensione originale utilizzando il metodo `actualSize()`.



Il pulsante Adattamento ampiezza allarga l'immagine della pagina del file di spool verso i bordi sinistro e destro della frame del programma di visualizzazione utilizzando il metodo `fitWidth()`.



Il pulsante Adattamento della pagina allarga l'immagine della pagina del file di spool in modo verticale e

orizzontale per adattarsi alla frame del programma di visualizzazione file sottoposto a spool utilizzando il metodo fitPage().



Il pulsante Zoom consente di aumentare o diminuire la dimensione dell'immagine della pagina del file di spool selezionando una delle percentuali presenti o inserendo una percentuale a piacere nel campo di testo che appare nella finestra di dialogo dopo aver selezionato il pulsante zoom.



Il pulsante Andare alla pagina, se selezionato, consente di spostarsi su una pagina specifica all'interno del file di spool.



Il pulsante Prima pagina, se selezionato, riporta alla prima pagina del file di spool e indica che si è sulla prima pagina quando non è attivo.



Il pulsante Pagina precedente, se selezionato, riporta alla pagina che precede quella che si sta visualizzando.



Il pulsante Pagina successiva, se selezionato, consente di avanzare sulla pagina successiva a quella che si sta visualizzando.



Il pulsante Ultima pagina, se selezionato, consente di avanzare verso l'ultima pagina del file di spool e indica che ci si trova sull'ultima pagina quando non è attivo.



Il pulsante Caricamento della prima pagina, se selezionato, carica la pagina visualizzata in precedenza utilizzando il metodo loadFlashPage().



Il pulsante impostazione della dimensione foglio quando viene selezionato consente di impostare la dimensione del foglio.



Il pulsante di impostazione risoluzione della visualizzazione quando viene selezionato consente di impostare la risoluzione della visualizzazione.

# 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.
// Presupporre che "system" sia un oggetto
// AS400 creato e inizializzato da qualche
// altra parte. Il testo della voce
// menu cita "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 del server creati dal programma:

```
// 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 la lista dei messaggi server
        // generata dal programma.
        AS400Message[] messageList = sourceMenuItem.getMessageList
();

        // ... Elaborare la lista 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 convertire i 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
        // 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);

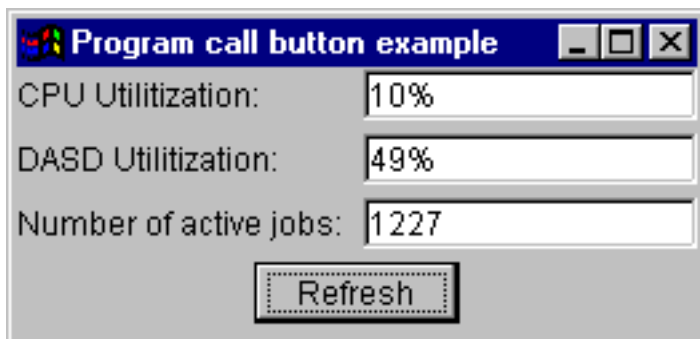
        // ... una volta chiamato il programma,
        // richiamare il valore restituito come
        // secondo parametro.
        int result = parm2Converter.toInt (parm2.getOutputData ());

```

## 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.

Sono disponibili i seguenti componenti:

- [RecordListFormPane](#) presenta una lista di record che derivano da un file del server in un modulo.
- [RecordListTablePane](#) presenta una lista di record che derivano da un un file del server in una tabella.
- [RecordListTableModel](#) gestisce la lista 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 [Specificare la 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.
        // Supponiamo che "system" sia un
        // oggetto AS400 creato e
        // inizializzato altrove. 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.
        // Presupporre 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 il [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.
        // Presupporre 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.
        // Presupporre 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**

 RecordListFormPane example

Record number: 1

CUSNUM

LSTNAM

INIT

STREET

CITY

STATE

ZIPCOD

CDTLMT

CHGCOD

BALDUE

CDTDUE



## 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.
        // Presupporre 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 RecordListModel

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 [RecordListModel](#) con JTable di JFC (Java Foundation Classes). La classe RecordListModel 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 RecordListModel e fornire un'integrazione personalizzata con un diverso componente GUI.

Per utilizzare un RecordListModel, 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 RecordListModel e lo presenta con un JTable:

```
        // Creare un oggetto RecordListModel.
        // Presupporre che "system" sia un oggetto
        // AS400 creato ed
        // inizializzato altrove.
RecordListModel tableModel = new RecordListModel (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 una lista delle risorse in una GUI.

- ResourceListPane visualizza il contenuto di una lista delle risorse in un javax.swing.JList grafico. Ogni voce visualizzata nella lista rappresenta un oggetto risorsa dalla lista delle risorse.
- ResourceListDetailsPane visualizza il contenuto della lista delle risorse in un javax.swing.JTable grafico. Ogni riga della tabella rappresenta un oggetto risorsa dalla lista 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: visualizzare una lista delle risorse in una GUI

Questo esempio crea un ResourceList di tutti gli utenti sul sistema e lo visualizza in una GUI (pannello dettagli):

```
// Creare la lista risorse.
AS400 system = new AS400("MYSYSTEM", "MYUSERID", "MYPASSWORD");
RUserList userList = new RUserList(system);

// Creare il ResourceListDetailsPane. In questo esempio,
// vi sono due colonne nella tabella. La prima colonna
// contiene le icone ed i nomi per ogni utente. La
// 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 la lista
// di utenti viene richiamata dall'iSeries.
detailsPane.load ();
```



# Classi Stato del sistema

I componenti System Status nel pacchetto vaccess consentono di creare le GUI utilizzando [AS400Panels](#) esistenti. Inoltre è disponibile l'opzione che consente di creare GUI proprie utilizzando le JFC (Java Foundation Classes). L'oggetto [VSystemStatus](#) 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 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 [AS400Panels](#) esistenti o creando pannelli propri tramite le JFC (Java Foundation Classes). L'oggetto [VSystemValueList](#) rappresenta una lista 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

L'esempio seguente crea una GUI dei valori di sistema utilizzando il Pannello AS400Explorer:

```
//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 le liste degli utenti del server e dei gruppi attraverso la [Classe VUser](#).

Sono disponibili i seguenti componenti:

- [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 una lista 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 della lista. 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. La lista degli utenti e dei gruppi non è stata caricata. 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 della lista.

Al momento dell'esecuzione, fare clic con il tastino destro del mouse su un utente, una lista 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 una lista 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.
- Lista utente - Gestire le proprietà delle informazioni sull'utente e sul gruppo. E' inoltre possibile modificare il contenuto della lista.
- 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 ();

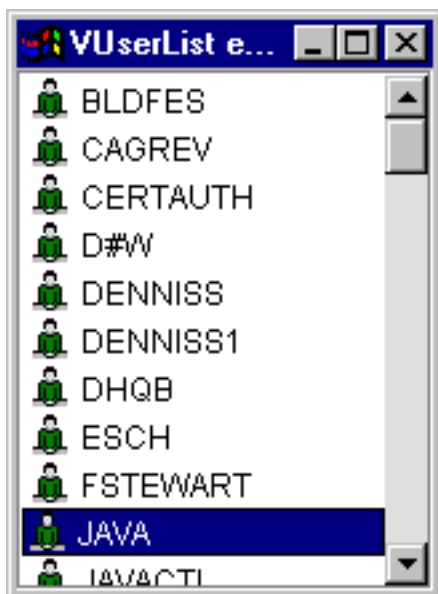
// Aggiungere il pannello explorer ad una frame
// Supponiamo che "frame" sia un JFrame creato altrove
frame.getContentPane().add(explorerPane);

```

## Altri esempi

Presentare una lista 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.

## 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.

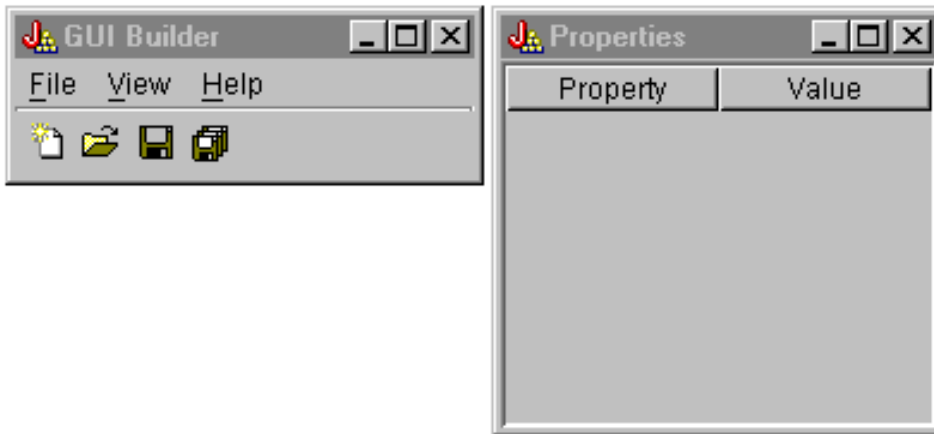
## Dettagli di Graphical Toolbox

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

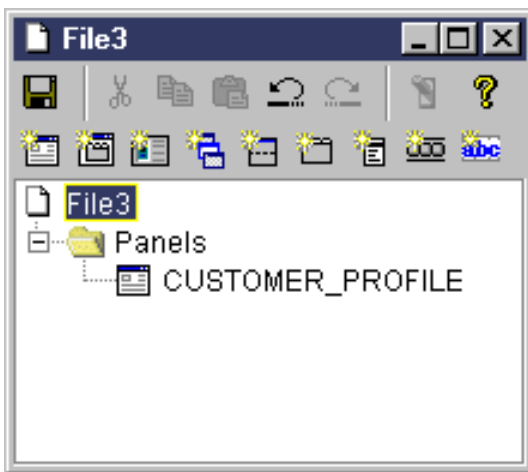
Quando si richiama per la prima volta il GUI Builder, vengono visualizzate due finestre, come mostrato nella Figura 1:

**Figura 1: finestre del GUI Builder**



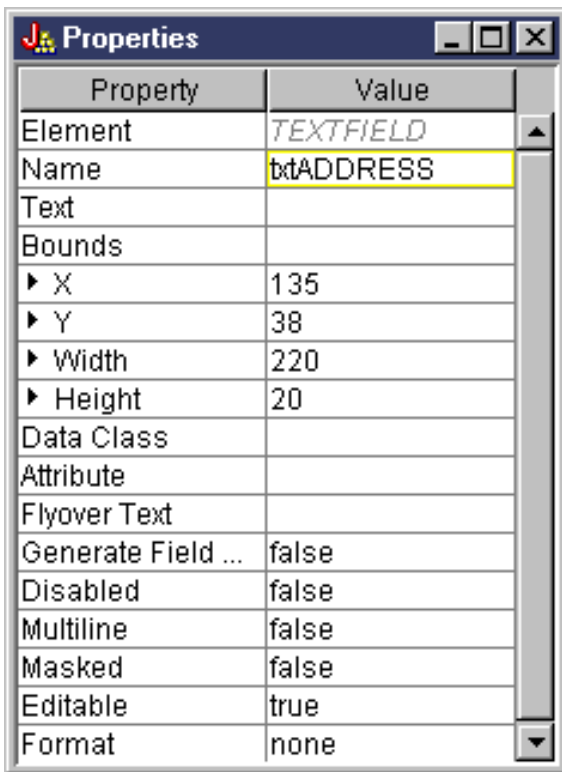
Utilizzare la [Finestra del programma di creazione del file](#) per creare e modificare i file PDML.

**Figura 2: finestra del programma di creazione del file**



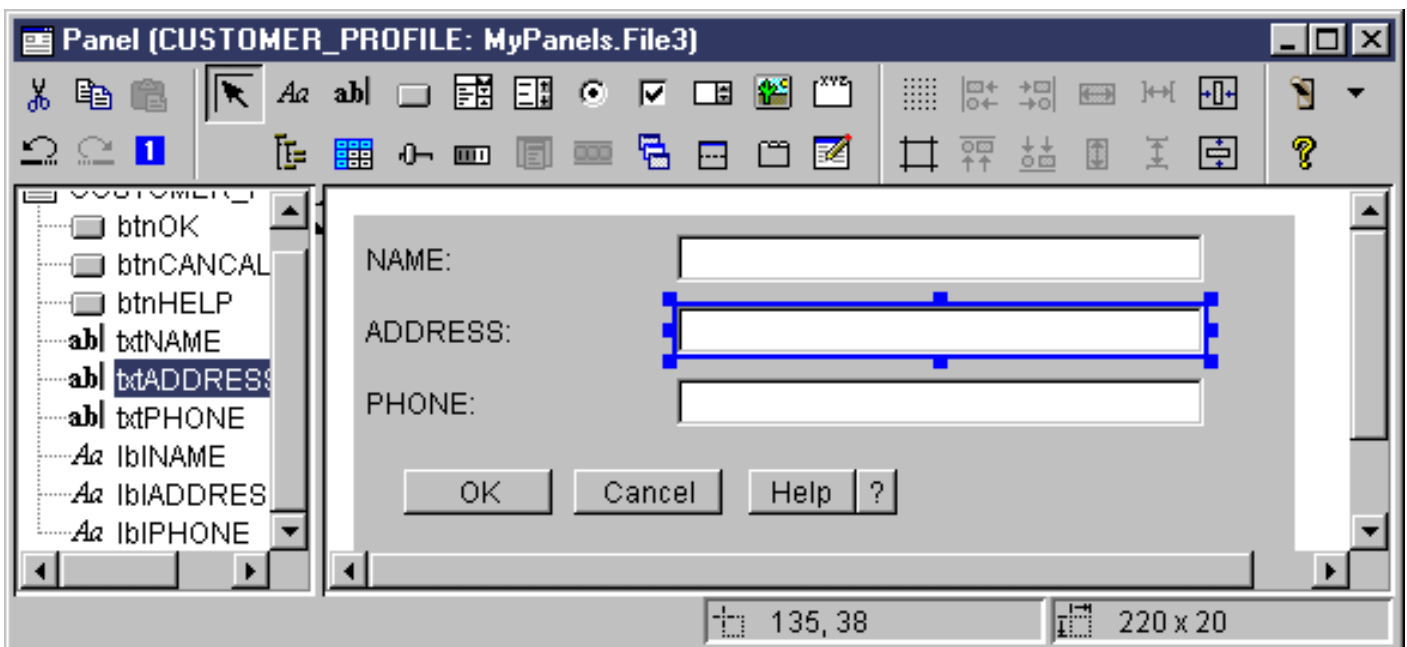
Utilizzare la [finestra Proprietà](#) per visualizzare o modificare le proprietà del controllo attualmente selezionato.

**Figura 3: finestra Proprietà**



Utilizzare la [finestra del 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.

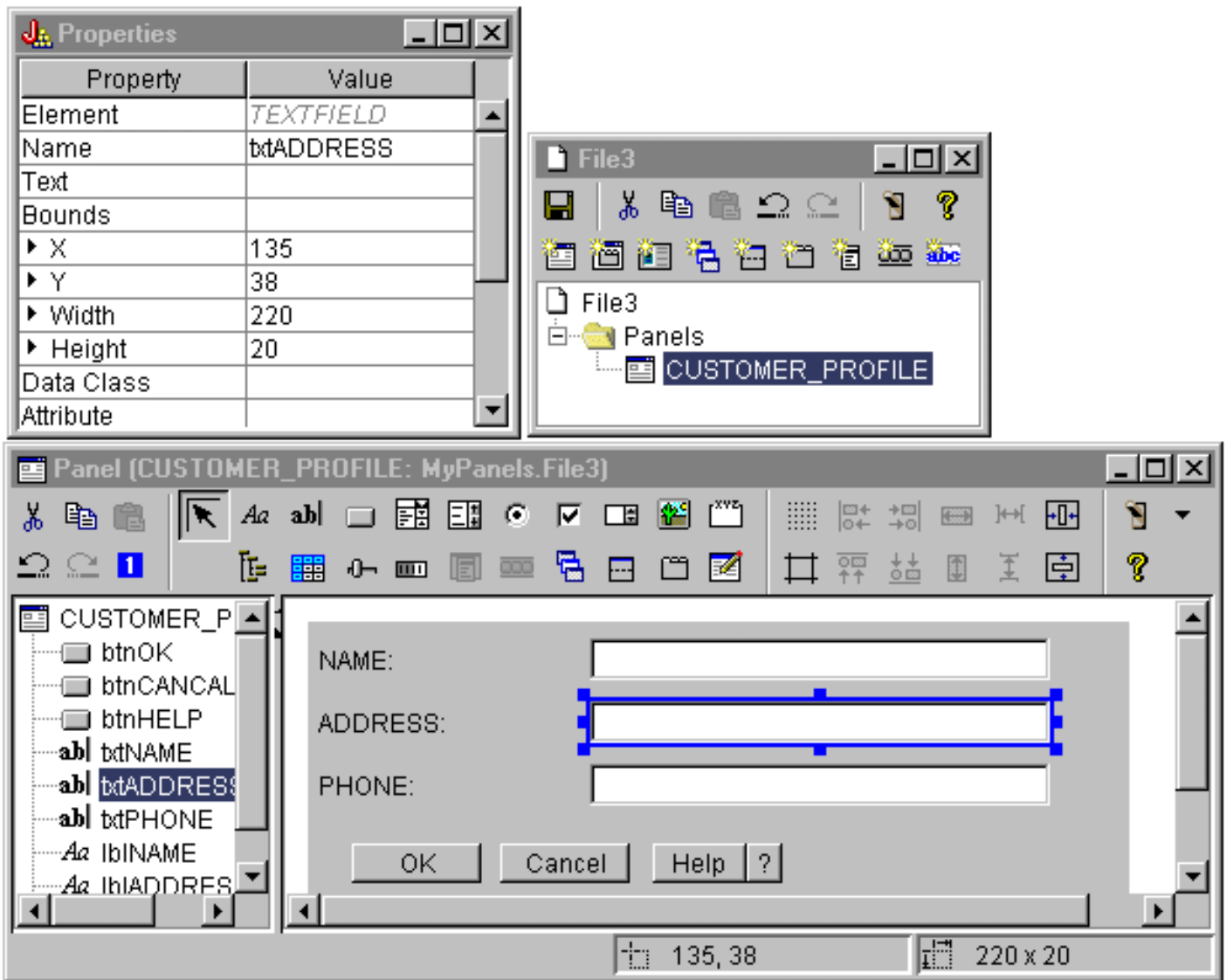
**Figura 4: finestra del programma di creazione del pannello**



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**

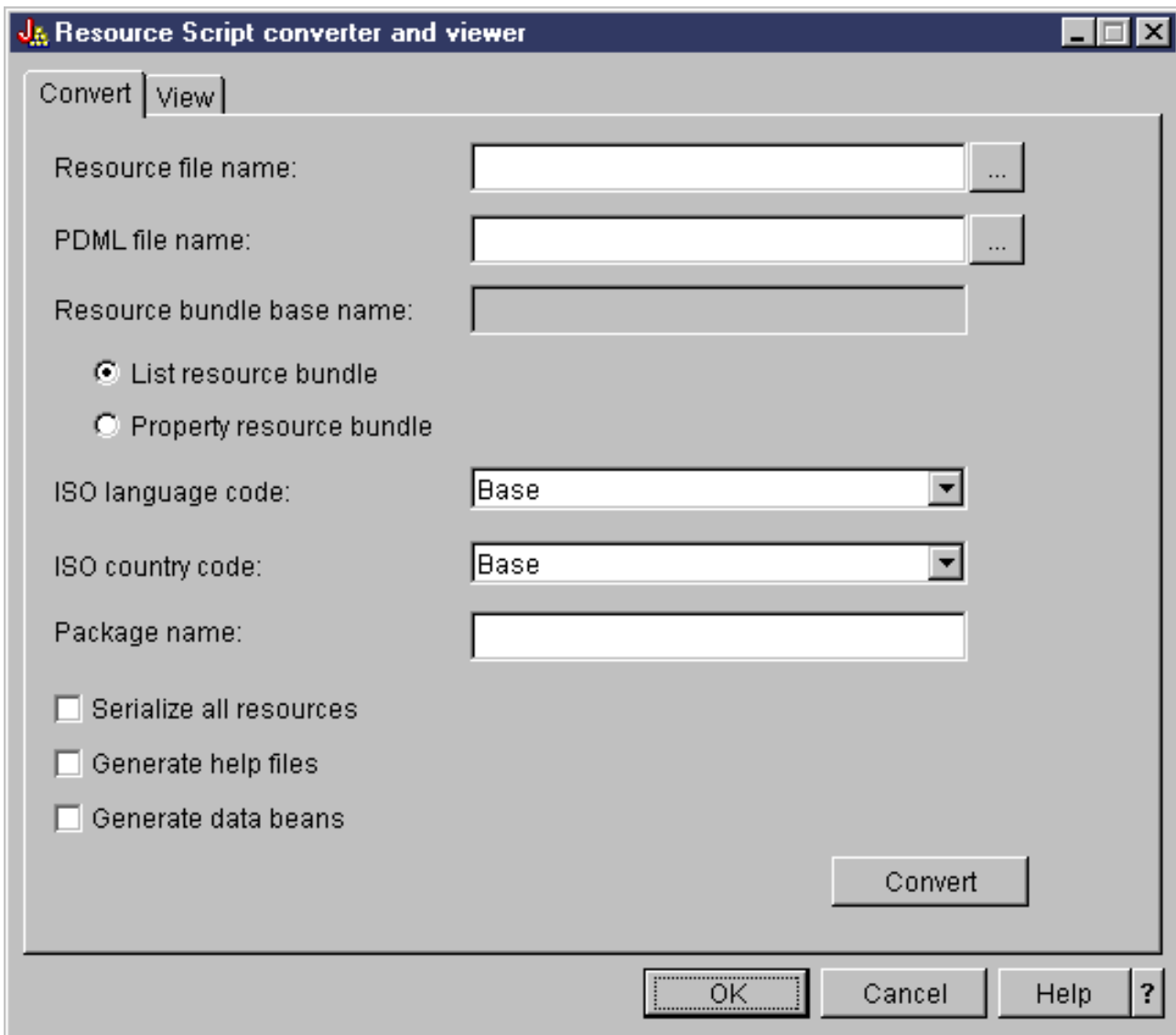




## Resource Script Converter

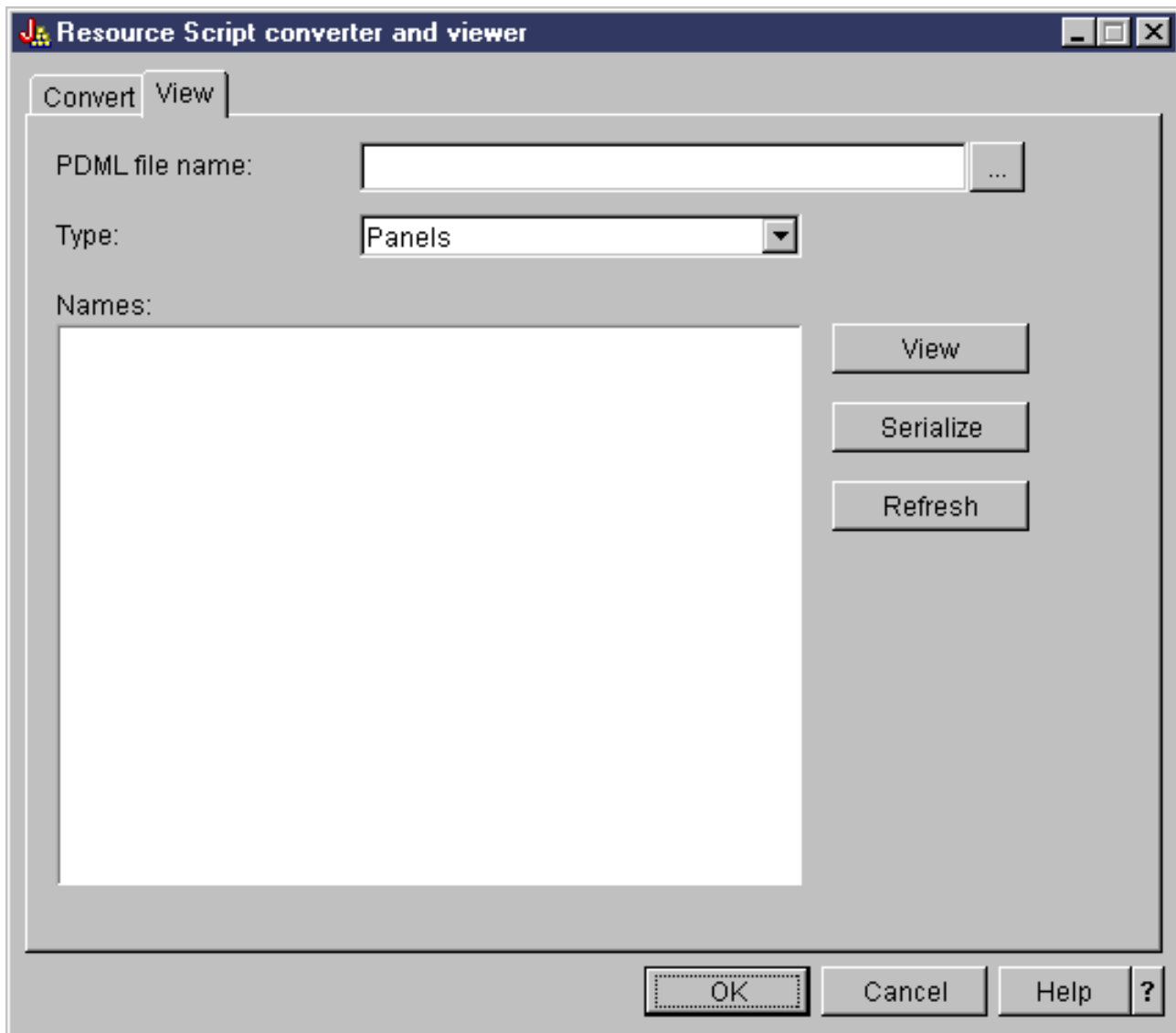
Il [Resource Script Converter](#) è composto da una finestra di dialogo con separatori a due pannelli. Nel pannello **Conversione** si specifica il nome del file RC Microsoft o VisualAge per Windows che è necessario convertire in PDML. E' possibile specificare il nome del file PDML di destinazione e 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 appena 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:

- [Installare il Graphical Toolbox](#)
- [Creare l'interfaccia utente](#)
- [Visualizzare i pannelli durante il tempo di esecuzione](#)
- [Creare i file dell'aiuto in linea](#)
- [Esempio del Graphical Toolbox](#)
- [Utilizzare il 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](#).

## Installare il 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:

### Trasferire i file JAR

**Nota:** la lista 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 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](#).

### Installare il 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.

## Impostare il percorso classe

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\x4j400.jar;  
C:\gtbox\lib\jhall.jar;
```

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 percorso presenti nel proprio classpath dovrebbero riportare quanto segue.

## Descrivere i 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 questa 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 (Program Call Markup Language). 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 del programma di utilità per formattare i dati iSeries e gestire i messaggi iSeries. Tali classi possono essere ridistribuite con applicazioni.
- **x4j400.jar** Contiene il programma di analisi XML utilizzato dalle classi API per interpretare i documenti PDML e PCML.
- **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.

**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.) Questo file JAR supplementare dovrebbe essere aggiunto al proprio percorso classe prima di **uitools.jar** nell'ordine della ricerca.

## Utilizzare il 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](#)

# Creare l'interfaccia utente

## Eseguire il GUI Builder

Per avviare il GUI Builder, richiamare l'interprete Java nel modo seguente:

```
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 sulla 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 una lista 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. Si dovrebbe inserire il contenuto dell'aiuto nei punti appropriati nell'HTML, entro le tag `<!-- HELPDOC : SEGMENTBEGIN -->` e `<!-- HELPDOC : SEGMENTEND -->`. Per informazioni più specifiche sull'aiuto, consultare [Modificare i 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 getter/setter 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 getter/setter verranno richiamati per inserire i dati relativi al pannello.

**Nota:** il numero ed il tipo dei metodi getter/setter dipendono dal tipo di controllo UI a cui i metodi sono associati. 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 molto 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 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, semplicemente rinominare PANEL1 nel FILE2 e quindi aggiornare l'evento relativo alla condizionale nel FILE1 in modo che effettui questa modifica.

## Eeguire il 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 [Installare 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. **Options**

`-x name`

Il nome del file PDML creato. Ha come valore predefinito il nome del file RC che deve essere elaborato.

`-p name`

Il nome del file PROPERTIES creato. Ha come valore predefinito il nome del file PDML.

`-r name`

Il nome della sottoclasse ListResourceBundle creata. Ha come valore predefinito il nome del file PDML.

`-package name`

Il nome del pacchetto al quale verranno assegnate le risorse create. Se non viene 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 sono 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.

## Mettere in corrispondenza le 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 getter e setter per la proprietà **SampleAttribute**.

# Visualizzare i 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.

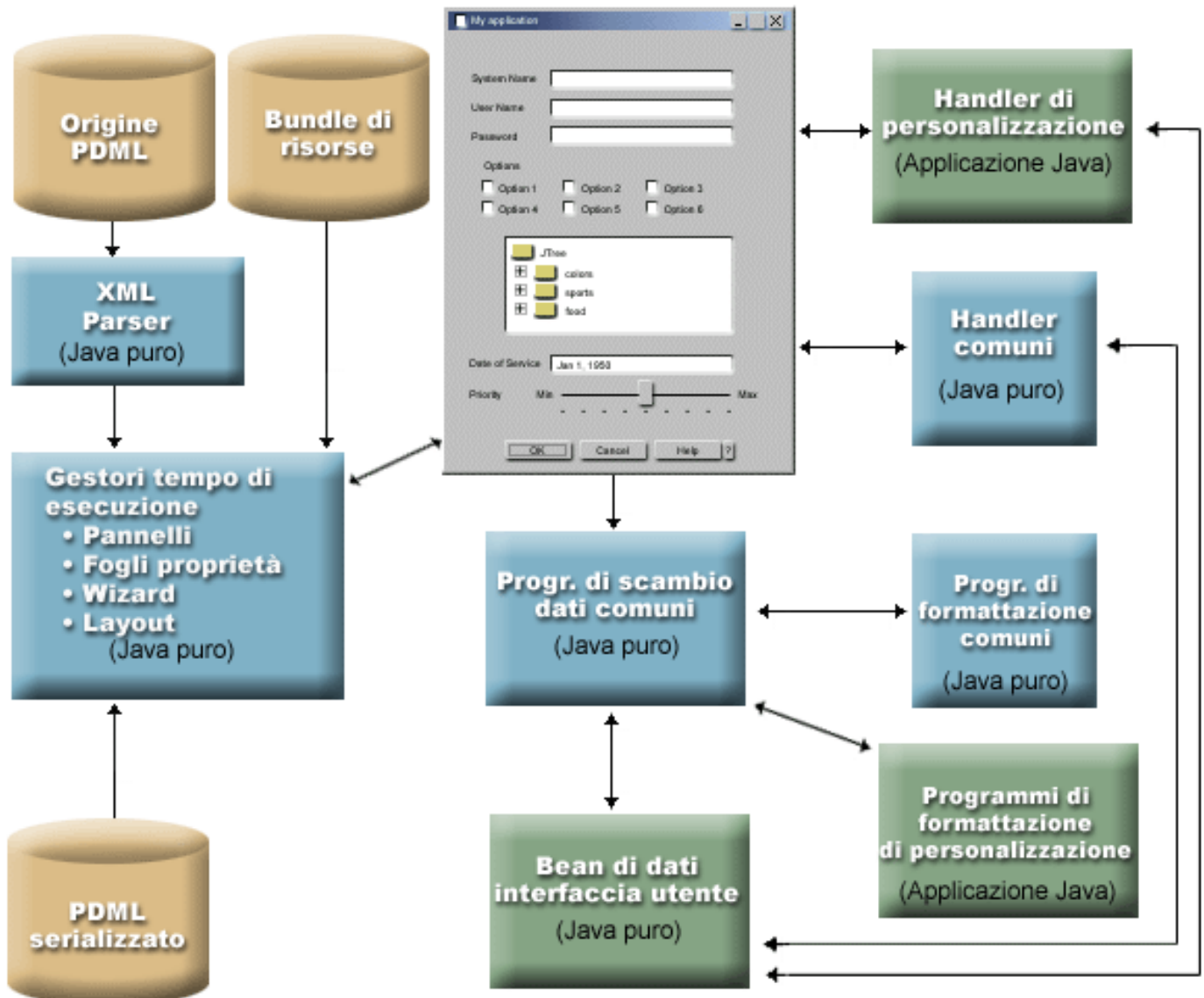
L'ambiente del tempo di esecuzione di Graphical Toolbox fornisce i seguenti servizi:

- 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](http://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.

### Esempio: creare e visualizzare 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. Lista 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: creare 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. Lista 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: utilizzare il 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 pannello dinamico. Parametri:
// 1. Nome risorsa della definizione di pannello
// 2. Nome del pannello
// 3. Lista 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.

# Modificare i 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 --> e <!-- 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 controlla quanto strettamente un segmento dipenda dai controlli definiti in PDML. Se PDMLSYNCH è impostato su "YES", il segmento del documento di aiuto viene rimosso se il controllo con lo stesso nome viene rimosso nel PDML. PDMLSYNCH="NO" indica che l'argomento deve rimanere nel documento di aiuto a prescindere dall'esistenza di un controllo corrispondente in PDML. Ciò viene utilizzato, ad esempio, quando si creano degli argomenti aggiuntivi o un argomento comune.

L'aiuto creato per un pannello ha dei collegamenti per ogni controllo abilitato per l'aiuto sul pannello. Questi collegamenti vengono creati con un riferimento ad ancora locale, in modo da poterli verificare come collegamenti interni in un browser standard. Quando il documento di aiuto viene diviso, il processore rimuove il cancelletto "#" da questi collegamenti interni trasformandoli in collegamenti esterni nei file HTML ad argomento singolo risultanti. Poiché potrebbero essere necessari dei collegamenti interni in un argomento, il processore rimuove solo i cancelletti "#" precedenti quando il riferimento dispone di un'estensione ".html" in esso incorporata.

Se si desidera sostituire l'intestazione predefinita per qualsiasi argomento particolare, 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>. Questo piè di pagina viene aggiunto ad ogni file HTML.
```

## Aggiungere i 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 in modo standard, 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 cancelletto "#" che lo precede.
- I collegamenti ad altri segmenti dell'argomento devono essere scritti con un "#" precedente, come se fossero un riferimento ad ancora interno.
- E' inoltre possibile creare collegamenti interni con altri segmenti di argomento. Durante il processo viene rimosso soltanto il cancelletto "#" iniziale.

### Note:

- 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.

## Apportare una modifica utilizzando un editor visuale

E' possibile modificare il contenuto dell'aiuto in quasi tutti gli editor HTML visuali. Poiché le tag HELPDOC sono commenti, potrebbero non essere evidenti in alcuni editor. Per comodità, è stata aggiunta un riga orizzontale alla struttura dell'aiuto prima della tag SEGMENTBEGIN e dopo la tag SEGMENTEND. Queste righe orizzontali forniscono un'indicazione visuale chiara dell'intero segmento in un editor visuale. Se si seleziona un segmento perché si desidera spostarlo, copiarlo o cancellarlo, è necessario selezionare le righe orizzontali adiacenti per verificare che siano state incluse le tag SEGMENTBEGIN e SEGMENTEND nella propria selezione. Queste righe orizzontali non vengono copiate sui singoli file HTML finali.



## Creare argomenti aggiuntivi

E' possibile creare segmenti di argomento aggiuntivi nel Documento di aiuto. Spesso è più semplice eseguire questa operazione copiando un altro segmento. Una volta copiato il segmento, è necessario copiare le righe orizzontali prima della tag `SEGMENTBEGIN` e dopo `SEGMENTEND`. Ciò renderà molto più semplice la modifica visuale e aiuterà ad evitare tag non corrispondenti. Per migliori risultati, utilizzare i suggerimenti che seguono:

- Il nome dell'ancora deve essere il nome che si desidera assegnare al file singolo risultante quando il documento di aiuto viene diviso. Deve terminare in ".html".
- E' necessario utilizzare la parola chiave `PDMLSYNCH="NO"` sulla tag `SEGMENTBEGIN` affinché il segmento non venga rimosso se viene creata la struttura dell'aiuto.
- Ogni riferimento al nuovo argomento sarà effettuato come un collegamento interno nel documento di aiuto, preceduto da un cancelletto "#". Il cancelletto "#" sarà rimosso dalle elaborazioni successive quando i segmenti saranno suddivisi in file singoli.

## Controllare i collegamenti

Per la maggior parte dei documenti scritti, è possibile controllare i collegamenti visualizzando il documento un browser web e selezionando diversi collegamenti. Nel documento di aiuto singolo, i collegamenti si trovano ancora nel 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 farlo tramite 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.



# Esempi di Graphical Toolbox

Vengono forniti degli esempi per mostrare le modalità di implementazione degli strumenti in Graphical Toolbox per i propri programmi UI.

- [Creare e visualizzare un pannello](#): mostra le modalità di creazione di un pannello semplice. L'esempio mostra quindi in che modo creare una piccola applicazione Java che visualizzi il pannello. Quando l'utente immette i dati nel campo del testo e fa clic sul pulsante Chiudi, l'applicazione replicherà i dati sulla console Java. Tale esempio illustra le funzioni di base e le operazioni dell'ambiente del Graphical Toolbox nel complesso.
- [Creare e visualizzare 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.
- [Creare una finestra di dialogo completamente funzionante](#): dopo aver implementato i DataBean che forniscono i dati al pannello ed aver identificato gli attributi nel PDML, questo esempio mostra in che modo creare una finestra di dialogo completamente funzionante.
- [Dimensionare il pannello utilizzando il gestore pannelli dinamico](#): il gestore pannelli dinamico regola dinamicamente il pannello durante il tempo di esecuzione.
- [Casella combinata modificabile](#): mostra un esempio di codifica di bean dei dati per una casella combinata modificabile.

I seguenti esempi mostrano in che modo il GUI Builder consente di creare:

- [Pannelli](#): mostra in che modo creare un pannello di esempio ed un codice di bean di dati che esegua il pannello
- [Pannelli sovrapposti](#): mostra in che modo creare un pannello sovrapposto e l'aspetto finale di un pannello sovrapposto
- [Fogli delle proprietà](#): mostra in che modo creare un foglio delle proprietà ed il suo aspetto finale
- [Pannelli suddivisi](#): mostra in che modo creare un pannello suddiviso ed il suo aspetto finale
- [Pannelli con separatori](#): mostra in che modo creare un pannello con separatore ed il suo aspetto finale
- [Wizard](#): mostra in che modo creare un wizard e l'aspetto finale del prodotto
- [Barre degli strumenti](#): mostra in che modo creare una barra degli strumenti ed il suo aspetto finale
- [Barre dei menu](#): mostra in che modo creare una barra dei menu ed il suo aspetto finale
- [Aiuto](#): mostra in che modo viene creato un documento di aiuto e le modalità di suddivisione del documento di aiuto nelle pagine dell'argomento. Consultare anche [Modificare i 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**

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.

# 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 avere a che fare con le idiosincrasie di singoli browser, si consiglia di eseguire le applet utilizzando il Modulo aggiuntivo Java di Sun. Altrimenti, sarà necessario creare file JAR siglati per Netscape e file CAB siglati separati per Internet Explorer.

## Creare l'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. Qui di seguito si trova 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 " +
1.1.5
                System.getProperty("java.version") + " - requires
                    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. Lista 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**, si assegna al pannello dell'applet la sua giusta dimensione e si sovrappone il metodo **paint** per conservare la dimensione del pannello quando viene realizzata la finestra del browser.

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 raccomanda 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 interpretata 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 d'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>

<!-- BEGIN JAVA(TM) PLUG-IN APPLET TAGS -->

<!-- The following tags use a special syntax which allows both Netscape and Internet Explorer to
load -->
<!-- the Java Plug-in and run the applet in the Plug-in's JRE. Do not modify this syntax.
-->
<!-- For more information see
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/~user/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>

<!-- END JAVA(TM) PLUG-IN APPLETS -->

<p>
</body>
</html>

```

E' importante che le informazioni sulla versione vengano impostate per 1.1.3.

**Nota:** in quest'esempio, si è scelto di memorizzare il file JAR del programma di analisi XML, **x4j400.jar**, sul server web. 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](#).

## Installare ed eseguire l'applet

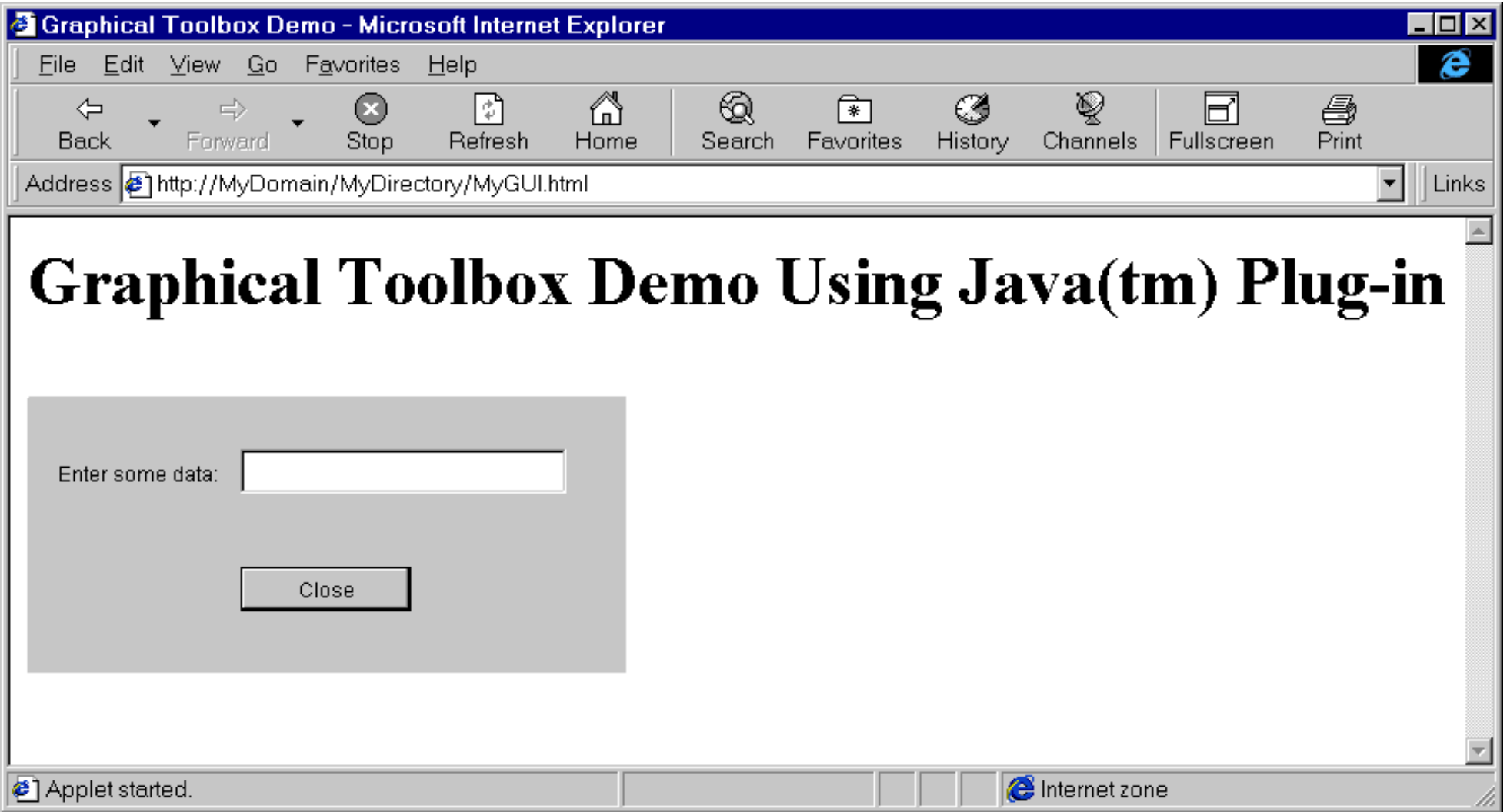
Installare l'applet sul server web preferito attenendosi alle seguenti istruzioni:

- Compilare **SampleApplet.java**.
- 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 **MyGUL.pdml** e del bundle di risorse **MyGUL.properties**.
- 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.
- Copiare i file JAR del Graphical Toolbox nell'indirizzario del server.
- Infine, copiare il file HTML **MyGUI.html** che contiene l'applet incorporata nell'indirizzario del server.

**Suggerimento:** durante il collaudo delle applet, assicurarsi di aver eliminato i 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:

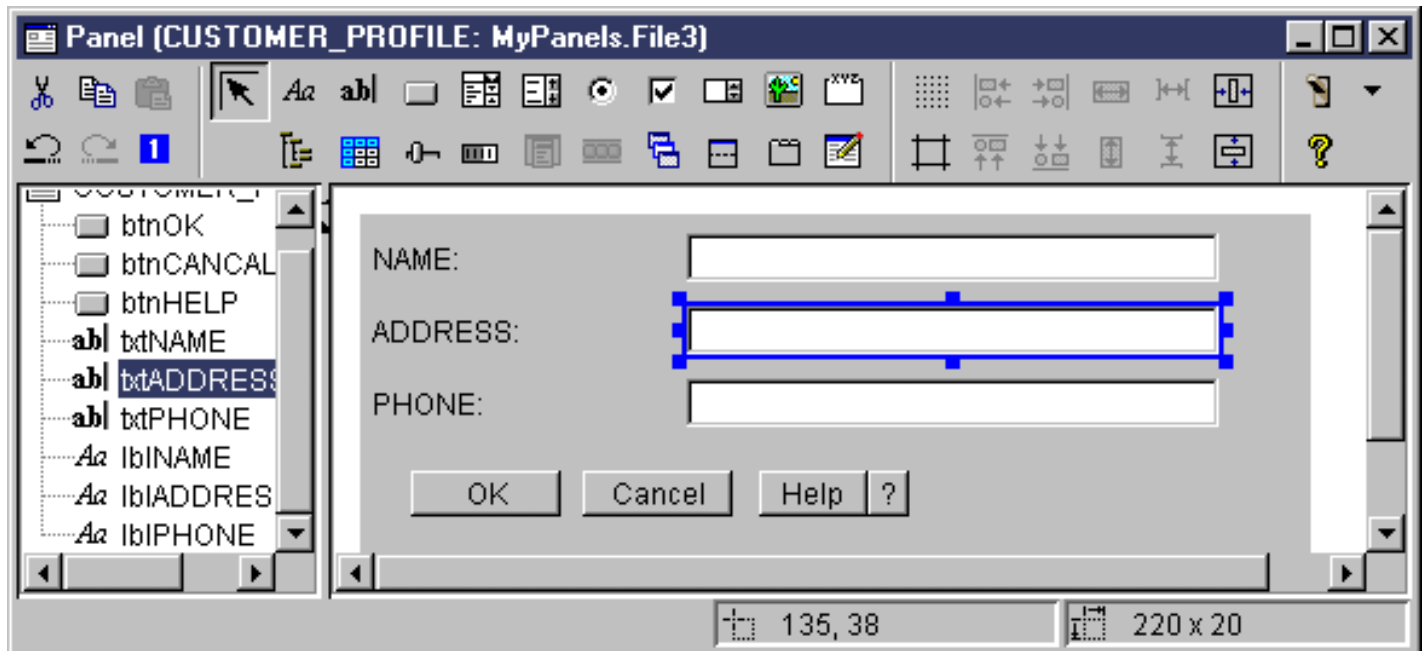
**Figura 1: esecuzione dell'applet di esempio in un browser**





# 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 una lista 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 Adatta altezza per adattare l'altezza di più componenti all'altezza di un componente specifico o primario.



Fare clic su Centra verticalmente per centrare verticalmente un componente selezionato nel 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 limite sinistro di un componente specifico o primario.



Fare clic su Allinea a destra per allineare più componenti su un pannello con il limite destro di un componente specifico o primario.



Fare clic su Adatta larghezza per adattare la larghezza di più componenti alla larghezza 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 ulteriori informazioni specifiche sul Graphical Toolbox.

# Bean IBM Toolbox per Java

JavaBeans<sup>(TM)</sup> 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

- Esempio: l'[esempio IBM Toolbox per Java](#) mostra un metodo di utilizzo dei JavaBean nel programma.
- Esempio: l'[esempio di codice del programma di creazione del bean visuale](#) mostra come creare un programma da JavaBean utilizzando un programma di creazione del bean visuale, ad esempio IBM Visual Age per Java.

# PMCL (Program Call Markup Language)

## Panoramica

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. Per ulteriori informazioni su XML, consultare la sezione [Riferimento XML](#).

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.

## Requisiti della piattaforma

Anche se PCML è stato progettato per supportare le chiamate del programma distribuito agli oggetti programma del server da una piattaforma Java, è possibile utilizzare PCML anche per effettuare chiamate ad un programma del server dall'interno dell'ambiente server.

## Argomenti per ulteriori informazioni

Fare riferimento ai seguenti argomenti su come utilizzare PCML:

- [Richiamare](#) i programmi con PCML
- Creare le chiamate al programma con le [tag](#) di PCML
- Un [esempio](#) di PCML

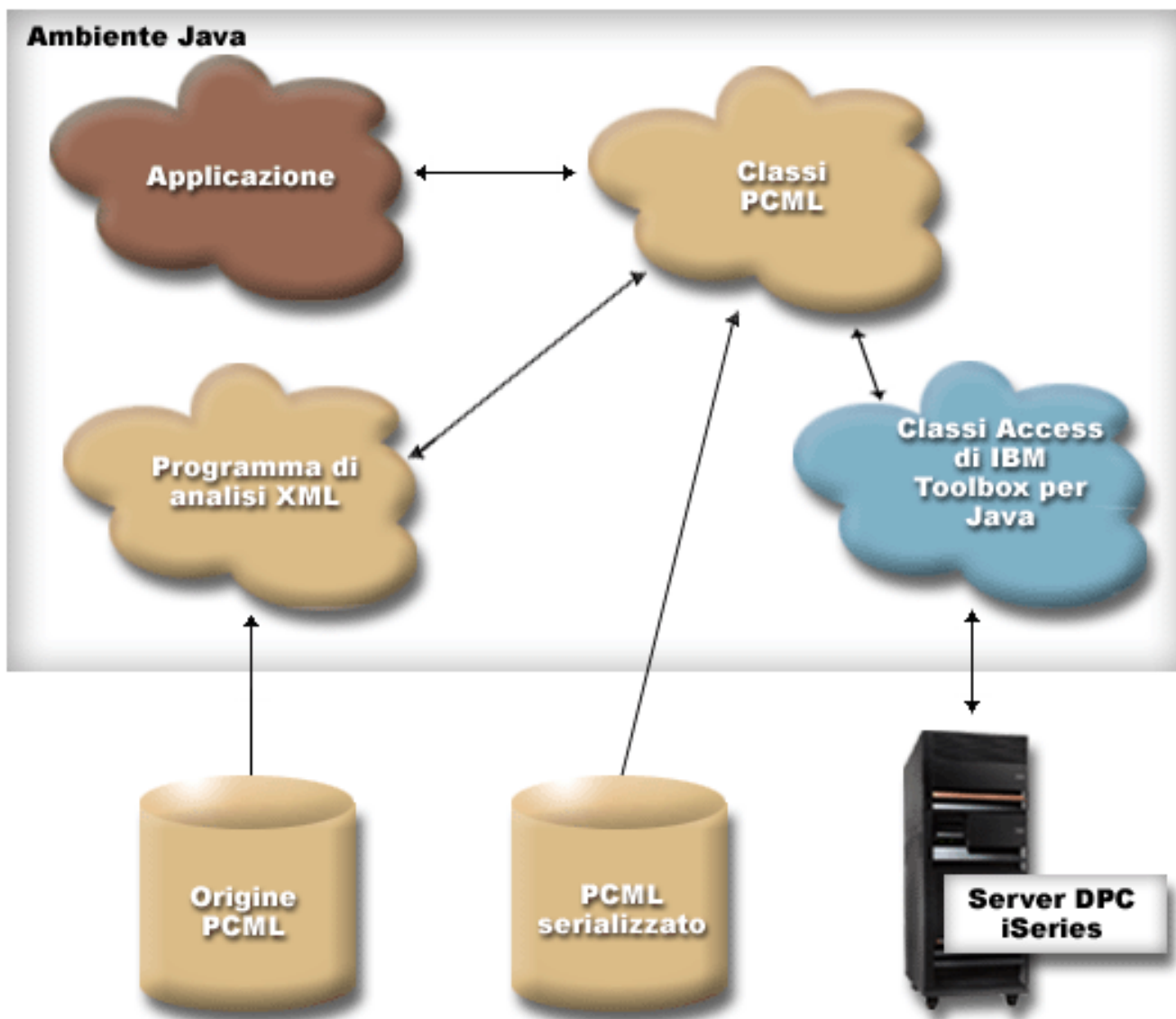
# Creare chiamate al programma iSeries con PCML

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 si descrivono le interfacce per i programmi iSeries 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 la propria applicazione crea l'oggetto ProgramCallDocument, IBM XML parser legge ed analizza il file sorgente PCML.

Una volta creata la classe ProgramCallDocument, il programma di applicazione utilizzerà i metodi della classe ProgramCallDocument per richiamare le informazioni necessarie dal server attraverso il server DPC (distributed program call) iSeries.

Per migliorare le prestazioni del tempo 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 di IBM non viene utilizzato durante il tempo di esecuzione. Fare riferimento a [Utilizzare file PCML serializzati](#).

## Utilizzare 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 CLASSPATH Java o il [metodo setPath\(\)](#) di ProgramCallDocument. Utilizzare il metodo setPath() quando il programma dell'applicazione Java deve impostare il percorso al file PCML durante il tempo 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");
```

## Utilizzare i file PCML serializzati

Per migliorare le prestazioni del tempo 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 consente all'utente migliori prestazioni in quanto, in fase di esecuzione, IBM XML parser 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">
    <!-- Parameter 1 contains a count of polygons along with an array of
polygons -->
    <struct name="parm1" usage="inputoutput">
      <data name="nbrPolygons" type="int" length="4" init="5" />
      <!-- Each polygon contains a count of the number of points along with
an array of points -->
      <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>
```

## Accedere 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);
    }
}

```

## Effettuare il 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 metodo che segue dalla classe [Trace](#) per attivare la traccia di 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, vengono sconsigliati in V5R2.

Il [metodo setFileName\(\)](#) di Trace consente di inviare i seguenti tipi di informazioni a specifici file di registrazione o, per impostazione predefinita, a System.out:«

- Un dump di dati esadecimale 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 questo formato dump:

```

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.

```

/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:

```





# Sintassi PCML

PCML è costituito dalle seguenti tag, ognuna delle quali ha tag di attributo proprie:

- La [tag program](#) inizia e chiude il codice che descrive un 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
<b>entrypoint=</b>	<i>entry-point-name</i>	Specifica il nome del punto d'immissione in un oggetto del programma di servizio che rappresenta la destinazione di questa chiamata al programma.
>> <b>epccsid=</b>	<i>ccsid</i>	Specifica il CCSID del punto d'immissione in un programma di servizio. Per ulteriori informazioni, consultare le note relative all'immissione del programma di servizio nel <a href="#">javadoc ServiceProgramCall</a> . <<
<b>name=</b>	<i>name</i>	Specifica il nome del programma.
<b>path=</b>	<i>path-name</i>	<p>Specifica il percorso per l'oggetto del programma. Il valore predefinito consiste nel presupporre che il programma si trovi nella libreria QSYS.</p> <p>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.</p> <p>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.</p> <p>Il nome percorso dovrebbe essere specificato con tutti i caratteri maiuscoli.</p> <p>&gt;&gt; Non utilizzare l'attributo <b>path</b> quando l'applicazione deve impostare il percorso al momento dell'esecuzione, ad esempio, quando l'utente specifica quale libreria viene utilizzata per l'installazione. In questo caso, utilizzare il <a href="#">metodo ProgramCallDocument.setPath()</a>. &lt;&lt;</p>

<b>parseorder=</b>	<i>name-list</i>	<p>Specifica l'ordine in cui verranno elaborati i parametri d'emissione. Il valore specificato è una lista di nomi parametro separata da spazi parametro in cui devono essere elaborati i parametri. I nomi nella lista devono essere identici ai nomi specificati nell'attributo <b>name</b> delle tag che appartengono a <b>&lt;program&gt;</b>. Il valore predefinito consiste nell'elaborare i parametri d'emissione nell'ordine in cui le tag appaiono nel documento.</p> <p>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.</p>
<b>returnvalue=</b>	<p><i>void</i> Il programma non restituisce un valore.</p> <p><i>integer</i> Il programma restituisce un valore intero con segno a 4 byte.</p>	<p>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.</p>
<b>threadsafe=</b>	<p><i>true</i> Il programma viene considerato protetto per il sottoprocesso.</p> <p><i>false</i> Il programma non è protetto per il sottoprocesso.</p>	<p>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 <i>true</i> produrrà prestazioni migliori.</p> <p>Per mantenere l'ambiente protetto, il valore predefinito consiste nel richiamare programmi in lavori server separati. Il valore predefinito è <i>false</i>.</p>

# 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
<b>name=</b>	<i>name</i>	Specificare il nome dell'elemento <b>&lt;struct&gt;</b>
<b>count=</b>	<i>number</i> dove <i>number</i> definisce una schiera fissa, immutabile.  <i>data-name</i> dove <i>data-name</i> definisce il nome di un elemento <b>&lt;data&gt;</b> nel documento PCML che conterrà, in fase di esecuzione, il numero di elementi nella schiera. E' possibile che il <i>data-name</i> specificato sia un nome completo o un nome relativo all'elemento corrente. In entrambi i casi, il nome deve fare riferimento ad un elemento <b>&lt;data&gt;</b> che è definito con <b>type="int"</b> . Consultare <a href="#">Risolvere i nomi relativi</a> per ulteriori informazioni sulle modalità di risoluzione dei nomi relativi.	Specifica se l'elemento è nella schiera e identifica il numero di voci nella schiera.  Se quest'attributo viene omissso, l'elemento non è definito come una schiera, anche se è possibile sia contenuto all'interno di un altro elemento definito come una schiera.
<b>maxvrm=</b>	<i>version-string</i>	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 <b>maxvrm</b> è utile per definire le interfacce di programma che differiscono tra i rilasci 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" stanno ad indicare una o più cifre rappresentanti rispettivamente la versione, il rilascio 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.

<b>minvrm=</b>	<i>version-string</i>	<p>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 rilasci di OS/400.</p> <p>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" stanno ad indicare una o più cifre rappresentanti rispettivamente la versione, il rilascio 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.</p>
<b>offset=</b>	<p><i>number</i> dove <i>number</i> definisce uno scostamento fisso, immutabile.</p> <p><i>data-name</i> dove <i>data-name</i> definisce il nome di un elemento &lt;data&gt; nel documento PCML che conterrà, in fase di esecuzione, lo scostamento rispetto a questo elemento. E' possibile che il <b>data-name</b> specificato sia un nome completo o un nome relativo all'elemento corrente. In entrambi i casi, è necessario che il nome faccia riferimento ad un elemento &lt;data&gt; definito con <b>type="int"</b>. Consultare <a href="#">Risolvere i nomi relativi</a> per ulteriori informazioni sulle modalità di risoluzione dei nomi relativi.</p>	<p>Specifica lo scostamento rispetto all'elemento &lt;struct&gt; in un parametro di emissione.</p> <p>Alcuni programmi restituiscono informazioni con una struttura fissa seguite da uno o più campi o strutture a lunghezza variabile. In questo caso, l'ubicazione di un elemento a lunghezza variabile è generalmente specificata come uno scostamento o spostamento nel parametro. L'attributo <b>offset</b> viene utilizzato per descrivere lo scostamento rispetto all'elemento &lt;struct&gt;.</p> <p><b>Offset</b> viene utilizzato insieme all'attributo <b>offsetfrom</b>. Se l'attributo <b>offsetfrom</b> non viene specificato, l'ubicazione di base per lo scostamento specificato nell'attributo <b>offset</b> è il principale dell'elemento. Consultare <a href="#">Specificare scostamenti</a> per ulteriori informazioni su come utilizzare gli attributi <b>offset</b> e <b>offsetfrom</b>.</p> <p>Gli attributi <b>offset</b> e <b>offsetfrom</b> 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.</p> <p>Se questo attributo viene omissso, l'ubicazione dei dati relativi a questo elemento viene immediatamente dopo il precedente elemento nel parametro, se presente.</p>
<b>offsetfrom=</b>	<p><i>number</i> dove <i>number</i> definisce un'ubicazione fissa, immutabile. Un attributo <i>number</i> è più tipicamente utilizzato per specificare <b>number="0"</b> che indica che lo scostamento è uno scostamento assoluto dall'inizio del parametro.</p> <p><i>data-name</i> dove <i>data-name</i> definisce il nome di un elemento &lt;data&gt; da utilizzare come una 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 <b>offset</b> sarà relativo all'ubicazione dell'elemento specificato su questo attributo. E' possibile che <i>data-name</i></p>	<p>Specifica l'ubicazione di base alla quale si riferisce l'attributo <b>offset</b>.</p> <p>Se l'attributo <b>offsetfrom</b> non è specificato, l'ubicazione di base per lo scostamento specificato sull'attributo <b>offset</b> è l'origine dell'elemento. Consultare <a href="#">Specificare scostamenti</a> per ulteriori informazioni su come utilizzare gli attributi <b>offset</b> e <b>offsetfrom</b> .</p> <p>Gli attributi <b>offset</b> e <b>offsetfrom</b> 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.</p>

	<p>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 <a href="#">Risolvere i nomi relativi</a> per ulteriori informazioni sulle modalità di risoluzione dei nomi relativi.</p> <p><i>struct-name</i> dove <i>struct-name</i> definisce il nome di un elemento <b>&lt;struct&gt;</b> 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 <b>offset</b> sarà relativo all'ubicazione dell'elemento specificato su questo attributo. Lo <i>struct-name</i> 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 <a href="#">Risolvere i nomi relativi</a> per ulteriori informazioni sulle modalità di risoluzione dei nomi relativi.</p>	
<b>outputsize=</b>	<p><i>number</i> dove <i>number</i> definisce un numero di byte da riservare fisso, immutato.</p> <p><i>data-name</i> dove <i>data-name</i> definisce il nome di un elemento <b>&lt;data&gt;</b> nel documento PCML che conterrà, in fase di esecuzione, il numero di byte da riservare per i dati di emissione. E' possibile che <i>data-name</i> specificato sia un nome completo o un nome relativo all'elemento corrente. In entrambi i casi, il nome deve fare riferimento ad un elemento <b>&lt;data&gt;</b> che è definito con <b>type="int"</b>. Consultare <a href="#">Risolvere i nomi relativi</a> per ulteriori informazioni sulle modalità di risoluzione dei nomi relativi.</p>	<p>Specifica il numero di byte da riservare per i dati di emissione per l'elemento. Per i parametri di emissione che hanno lunghezza variabile, è necessario l'attributo <b>outputsize</b> per specificare quanti byte dovrebbero essere riservati per i dati che devono essere restituiti dal programma server. <b>Outputsize</b> 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.</p> <p><b>Outputsize</b> non è necessario e non dovrebbe essere specificato per parametri di emissione a dimensione fissa.</p> <p>Il valore specificato sull'attributo è utilizzato come dimensione totale per l'elemento che include tutti i secondari dell'elemento. Perciò, l'attributo <b>outputsize</b> viene ignorato su ogni secondario o discendente dell'elemento.</p> <p>Se l'attributo viene omissso, il numero di byte da riservare per i dati di emissione è determinato al tempo di esecuzione aggiungendo il numero di byte da riservare per tutti i secondari dell'elemento <b>&lt;struct&gt;</b>.</p>
<b>usage=</b>	<p><i>inherit</i></p> <p><i>input</i></p> <p><i>output</i></p>	<p>L'utilizzo viene ereditato dall'elemento principale. Se la struttura non ha una struttura principale, si presume che l'utilizzo sia <b>inputoutput</b>.</p> <p>La struttura rappresenta un valore di immissione al programma host. Per i tipi numerici e di carattere, viene eseguita l'appropriata conversione.</p> <p>La struttura rappresenta un valore di emissione dal programma host. Per i tipi numerici e di carattere, viene eseguita l'appropriata conversione.</p>

## Specificare gli scostamenti

Alcuni programmi restituiscono informazioni con una struttura fissa seguite da uno o più campi o strutture a lunghezza variabile. In questo caso, l'ubicazione di un elemento a lunghezza variabile è generalmente specificata come uno scostamento o spostamento nel 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, dal momento che la distanza è 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">
    <!-- receiver variable contains a path -->
    <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">
    <!-- receiver variable contains an object -->
    <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 una lista 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=	<p><i>char</i> dove <i>char</i> indica un valore di carattere. Un valore dati <i>char</i> viene restituito come un <i>java.lang.String</i>. Per ulteriori informazioni, consultare i <a href="#">valori char per length</a>.</p> <p><i>int</i> dove <i>int</i> è un valore di numero intero. Un valore dati <i>int</i> viene restituito come un <i>java.lang.Long</i>. Per ulteriori informazioni, consultare i <a href="#">valori int per length e precision</a>.</p> <p><i>packed</i> dove <i>packed</i> è un valore decimale compresso. Un valore dati <i>packed</i> viene restituito come un <i>java.math.BigDecimal</i>. Per ulteriori informazioni, consultare i <a href="#">valori packed per length e precision</a>.</p> <p><i>zoned</i> dove <i>zoned</i> è un valore a zonatura decimale. Un valore dati <i>zoned</i> viene restituito come un <i>java.math.BigDecimal</i>. Per ulteriori informazioni, consultare i <a href="#">valori zoned per length e precision</a>.</p> <p><i>float</i></p>	<p>Indica il tipo di dati che si sta utilizzando (char, integer, packed, zoned, floating point, byte o struct).</p> <p>I valori degli attributi length e precision sono differenti per diversi tipi di dati. Per ulteriori informazioni, consultare <a href="#">Valori per length e precision</a>.</p>



dove *float* è 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*

dove *byte* è un valore di 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* specifica 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*.

**bidistringtype=**

*DEFAULT*

dove *DEFAULT* è [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](#).

Specifica il tipo stringhe bidirezionale per gli elementi **<data>** con **type="char"**. Se questo attributo viene omissso, 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 in [Javadoc per la classe BidiStringType](#).

<b>ccsid=</b>	<p><i>number</i> dove <i>number</i> definisce un CCSID fisso, immutato.</p> <p><i>data-name</i> dove <i>data-name</i> definisce il nome che conterrà, nel tempo di esecuzione, il CCSID dei dati carattere. E' possibile che <i>data-name</i> specificato sia un nome completo o un nome relativo all'elemento corrente. In entrambi i casi, è necessario che il nome faccia riferimento ad un elemento <b>&lt;data&gt;</b> definito con <b>type="int"</b>. Consultare <a href="#">Risolvere i nomi relativi</a> per ulteriori informazioni sulle modalità di risoluzione dei nomi relativi.</p>	<p>Specifica il CCSID (Coded Character Set ID) dell'host per i dati carattere per l'elemento <b>&lt;data&gt;</b>. L'attributo <b>ccsid</b> può essere specificato solo per gli elementi <b>&lt;data&gt;</b> con <b>type="char"</b>.</p> <p>Se questo attributo viene omissso, si presume che i dati carattere per questo elemento siano nel CCSID predefinito dell'ambiente host.</p>
<b>▶chartype=</b>	<p><i>onebyte</i> dove <i>onebyte</i> specifica la dimensione di ogni carattere.</p> <p><i>twobyte</i> dove <i>twobyte</i> specifica la dimensione di ogni carattere.</p> <p>Quando si utilizza <i>chartype</i>, l'attributo <b>length="number"</b> specifica il numero di caratteri, non il numero di byte.</p>	<p>Specifica la dimensione di ogni carattere. ◀</p>
<b>count=</b>	<p><i>number</i> dove <i>number</i> definisce un numero di elementi fissi, immutati in una schiera dimensionata.</p> <p><i>data-name</i> dove <i>data-name</i> definisce il nome di un elemento <b>&lt;data&gt;</b> nel documento PCML che conterrà, in fase di esecuzione, il numero di elementi nella schiera. E' possibile che il <i>data-name</i> specificato sia un nome completo o un nome relativo all'elemento corrente. In entrambi i casi, il nome deve fare riferimento ad un elemento <b>&lt;data&gt;</b> che è definito con <b>type="int"</b>. Consultare <a href="#">Risolvere i nomi relativi</a> per ulteriori informazioni sulle modalità di risoluzione dei nomi relativi.</p>	<p>Specifica se l'elemento è nella schiera e identifica il numero di voci nella schiera.</p> <p>Se l'attributo <i>count</i> viene omissso, 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.</p>
<b>init=</b>	<p><i>string</i></p>	<p>Specifica un valore iniziale per l'elemento <b>&lt;data&gt;</b>. Il valore <i>init</i> viene utilizzato se un valore iniziale non è impostato esplicitamente dal programma di applicazione quando gli elementi <b>&lt;data&gt;</b> vengono utilizzati con <b>usage="input"</b> o <b>usage="inputoutput"</b>.</p> <p>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.</p>

<b>length=</b>	<p>» <i>number</i> dove <i>number</i> definisce il numero di byte richiesti dai dati. Tuttavia, quando si utilizza l'attributo <i>chartype</i>, <i>number</i> specifica il numero di caratteri, non il numero di byte.◀◀</p> <p><i>data-name</i> dove <i>data-name</i> definisce il nome di un elemento &lt;data&gt; nel documento PCML che conterrà, in fase di esecuzione, la lunghezza. Un <i>data-name</i> può essere specificato solo per gli elementi &lt;data&gt; con <b>type="char"</b> o <b>type="byte"</b>. E' possibile che <i>data-name</i> specificato sia un nome completo o un nome relativo all'elemento corrente. In entrambi i casi, è necessario che il nome faccia riferimento ad un elemento &lt;data&gt; definito con <b>type="int"</b>. Consultare <a href="#">Risolvere i nomi relativi</a> per ulteriori informazioni sulle modalità di risoluzione dei nomi relativi.</p>	<p>Specifica la lunghezza dell'elemento data. La variazione nell'utilizzo di questo attributo dipende dal tipo di dati. Per ulteriori informazioni, consultare i valori <a href="#">per length e precision</a>.</p>
<b>maxvrm=</b>	<i>version-string</i>	<p>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 rilasci di iSeries.</p> <p>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 rispettivamente la versione, il livello di rilascio 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.</p>
<b>minvrm=</b>	<i>version-string</i>	<p>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 rilasci di iSeries.</p> <p>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" stanno ad indicare una o più cifre rappresentanti rispettivamente la versione, il rilascio 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.</p>
<b>name=</b>	<i>name</i>	<p>Specifica il nome dell'elemento &lt;data&gt;.</p>

<b>offset=</b>	<p><i>number</i> dove <i>number</i> definisce uno scostamento fisso, immutabile.</p> <p><i>data-name</i> dove <i>data-name</i> definisce il nome di un elemento <b>&lt;data&gt;</b> nel documento PCML che conterrà, in fase di esecuzione, lo scostamento rispetto a questo elemento. E' possibile che il <i>data-name</i> specificato sia un nome completo o un nome relativo all'elemento corrente. In entrambi i casi, il nome deve fare riferimento ad un elemento <b>&lt;data&gt;</b> che è definito con <b>type="int"</b>. Consultare <a href="#">Risolvere i nomi relativi</a> per ulteriori informazioni sulle modalità di risoluzione dei nomi relativi.</p>	<p>Specifica lo scostamento all'elemento <b>&lt;data&gt;</b> in un parametro di emissione.</p> <p>Alcuni programmi restituiscono informazioni con una struttura fissa seguite da uno o più campi o strutture a lunghezza variabile. In questo caso, l'ubicazione di un elemento a lunghezza variabile è generalmente specificata come uno scostamento o spostamento nel parametro.</p> <p>Un attributo <b>offset</b> viene utilizzato in congiunzione con l'attributo <b>offsetfrom</b>. Se l'attributo <b>offsetfrom</b> non viene specificato, l'ubicazione di base per lo scostamento specificato nell'attributo <b>offset</b> è il principale dell'elemento. Consultare <a href="#">Specificare scostamenti</a> per ulteriori informazioni su come utilizzare gli attributi <b>offset</b> e <b>offsetfrom</b>.</p> <p>Gli attributi <b>offset</b> e <b>offsetfrom</b> 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.</p> <p>Se questo attributo viene omesso, l'ubicazione dei dati relativi a questo elemento viene immediatamente dopo il precedente elemento nel parametro, se presente.</p>
<b>offsetfrom=</b>	<p><i>number</i> dove <i>number</i> definisce una ubicazione base fissa, immutabile. <i>Number</i> è più tipicamente utilizzato per specificare <b>number="0"</b> che indica che lo scostamento è uno scostamento assoluto dall'inizio del parametro.</p> <p><i>data-name</i> dove <i>data-name</i> definisce il nome di un elemento <b>&lt;data&gt;</b> utilizzato come una ubicazione base per lo scostamento. Il nome dell'elemento specificato deve essere il principale o l'origine di questo elemento. Il valore dall'attributo <b>offset</b> sarà relativo all'ubicazione dell'elemento specificato su questo attributo. E' possibile che <i>data-name</i> 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 <a href="#">Risolvere i nomi relativi</a> per ulteriori informazioni sulle modalità di risoluzione dei nomi relativi.</p> <p><i>struct-name</i> dove <i>struct-name</i> definisce il nome di un elemento <b>&lt;struct&gt;</b> utilizzato come una 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 <b>offset</b> sarà relativo all'ubicazione dell'elemento specificato su</p>	<p>Specifica l'ubicazione di base alla quale si riferisce l'attributo <b>offset</b>.</p> <p>Se l'attributo <b>offsetfrom</b> non è specificato, l'ubicazione di base per lo scostamento specificato sull'attributo <b>offset</b> è l'origine dell'elemento. Consultare <a href="#">Specificare scostamenti</a> per ulteriori informazioni su come utilizzare gli attributi <b>offset</b> e <b>offsetfrom</b> .</p> <p>Gli attributi <b>offset</b> e <b>offsetfrom</b> 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.</p>

	<p>questo attributo. Lo <i>struct-name</i> 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 <a href="#">Risolvere i nomi relativi</a> per ulteriori informazioni sulle modalità di risoluzione dei nomi relativi.</p>	
<b>outputsize=</b>	<p><i>number</i> dove un <i>number</i> definisce un numero fisso, imm modificabile di byte da riservare.</p> <p><i>data-name</i> dove <i>data-name</i> definisce il nome di un elemento <b>&lt;data&gt;</b> nel documento PCML che conterrà, in fase di esecuzione, il numero di byte da riservare per i dati di emissione. E' possibile che <i>data-name</i> specificato sia un nome completo o un nome relativo all'elemento corrente. In entrambi i casi, il nome deve fare riferimento ad un elemento <b>&lt;data&gt;</b> che è definito con <b>type="int"</b>. Consultare <a href="#">Risolvere i nomi relativi</a> per ulteriori informazioni sulle modalità di risoluzione dei nomi relativi.</p>	<p>Specifica il numero di byte da riservare per i dati di emissione per l'elemento. Per i parametri di emissione che hanno lunghezza variabile, è necessario l'attributo <b>outputsize</b> per specificare quanti byte dovrebbero essere riservati per i dati che devono essere restituiti dal programma iSeries. Un attributo <b>outputsize</b> 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.</p> <p><b>Outputsize</b> non è necessario e non dovrebbe essere specificato per parametri di emissione a dimensione fissa.</p> <p>Il valore specificato su questo attributo è utilizzato come dimensione totale per l'elemento che include tutti i secondari dell'elemento. Perciò, l'attributo <b>outputsize</b> viene ignorato su ogni secondario o discendente dell'elemento.</p> <p>Se <b>outputsize</b> viene omesso, il numero di byte da riservare per i dati di emissione è determinato in fase di esecuzione aggiungendo il numero di byte da riservare per tutti i secondari dell'elemento <b>&lt;struct&gt;</b>.</p>
<b>passby=</b>	<p><i>reference</i> dove <i>reference</i> indica che il parametro verrà passato per riferimento. Quando il programma viene richiamato, esso passerà un puntatore al valore di parametro.</p> <p><i>value</i> dove <i>value</i> indica un valore del numero intero. Questo valore è consentito solo quando <b>type="int"</b> e <b>length="4"</b> viene specificato.</p>	<p>Specifica se il parametro viene passato per riferimento o per valore. Questo attributo è consentito solo quando questo elemento è un secondario di un elemento <b>&lt;program&gt;</b> che definisce una chiamata al programma di servizio.</p>
<b>precision=</b>	<p><i>number</i></p>	<p>Specifica il numero di byte di precisione per alcuni tipi di dati numerici. Per ulteriori informazioni, consultare i valori <a href="#">per length e precision</a>.</p>
<b>struct=</b>	<p><i>name</i></p>	<p>specifica il nome di un elemento <b>&lt;struct&gt;</b> per l'elemento <b>&lt;data&gt;</b>. Un attributo <b>struct</b> può essere specificato solo per gli elementi <b>&lt;data&gt;</b> con <b>type="struct"</b>.</p>

<b>trim=</b>	<p><i>right</i> dove <i>right</i> è il funzionamento predefinito utilizzato per ridimensionare gli spazi bianchi finali.</p> <p><i>left</i> dove <i>left</i> è utilizzato per eliminare gli spazi bianchi iniziali.</p> <p><i>both</i> dove <i>both</i> è utilizzato per eliminare sia gli spazi bianchi finali che quelli iniziali.</p> <p><i>none</i> dove <i>none</i> indica che gli spazi bianchi non sono ridimensionati.</p>	Specifica come eliminare lo spazio bianco dai dati del carattere.◀
<b>usage=</b>	<i>inherit</i>	L'utilizzo viene ereditato dall'elemento principale. Se la struttura non ha una struttura principale, si presume che l'utilizzo sia <i>inputoutput</i> .
	<i>input</i>	Definisce un valore di immissione al programma host. Per i tipi numerici e di carattere, viene eseguita l'appropriata conversione.
	<i>output</i>	Definisce un valore di emissione dal programma host. Per i tipi numerici e di carattere, viene eseguita l'appropriata conversione.
	<i>inputoutput</i>	Definisce sia i valori di immissione che di emissione.

## Specificare gli scostamenti

Alcuni programmi restituiscono informazioni con una struttura fissa seguite da uno o più campi o strutture a lunghezza variabile. In questo caso, l'ubicazione di un elemento a lunghezza variabile è generalmente specificata come uno scostamento o spostamento nel 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, dal momento che la distanza è 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">
    <!-- receiver variable contains a path -->
    <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">
    <!-- receiver variable contains an object -->
    <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
<b>tipo</b> ="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 <i>number</i> costante letterale o un <i>data-name</i> .	Non applicabile
<b>tipo</b> ="int"	Il numero di byte di dati per questo elemento: 2, 4 o 8. E' necessario specificare un <i>number</i> costante letterale.	Indica il numero di bit di precisione e se il numero intero è con segno o senza segno: <ul style="list-style-type: none"> <li>● Per <b>length</b>="2" <ul style="list-style-type: none"> <li>○ Utilizzare <b>precision</b>="15" per un numero intero a 2 byte con segno. Questo è un valore predefinito</li> <li>○ Utilizzare <b>precision</b>="16" per un numero intero a 2 byte senza segno</li> </ul> </li> <li>● Per <b>length</b>="4" <ul style="list-style-type: none"> <li>○ Utilizzare <b>precision</b>="31" per un numero intero a 4 byte con segno</li> <li>○ Utilizzare <b>precision</b>="32" per un numero intero a 4 byte senza segno</li> </ul> </li> <li>● Per <b>length</b>="8" utilizzare <b>precision</b>="63" per un numero intero a 8 byte con segno</li> </ul>
<b>tipo</b> ="packed" o "zoned"	Il numero di cifre numeriche di dati per questo elemento. E' necessario specificare un <i>number</i> 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 <b>length</b> .
<b>tipo</b> ="float"	Il numero di byte, 4 o 8, di dati per questo elemento. E' necessario specificare un <i>number</i> costante letterale.	Non applicabile
<b>tipo</b> ="byte"	Il numero di byte di dati per questo elemento. E' necessario specificare un <i>number</i> costante letterale o <i>data-name</i> .	Non applicabile
<b>tipo</b> ="struct"	Non consentito.	Non applicabile

## Definire i 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 dalla tag <rfml>, « in tal caso il nome è considerato come nome assoluto e non



relativo.

Qui è presentato un esempio di utilizzo di PCML:

```
<pcml version="1.0">
  <program name="polytest" path="/QSYS.lib/MYLIB.lib/POLYTEST.pgm">
    <!-- Parameter 1 contains a count of polygons along with an array of
polygons -->
    <struct name="parml" usage="inputoutput">
      <data name="nbrPolygons" type="int" length="4" init="5" />
      <!-- Each polygon contains a count of the number of points along with
an array of points -->
      <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>
```

»Questo è un esempio di utilizzo di RFML:

```
<rfml version="4.0">
  <struct name="polygon">
    <!-- Each polygon contains a count of the number of points along with an
array of points. -->
    <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">
    <!-- This format contains a count of polygons along with an array of
polygons -->
    <data name="nbrPolygons" type="int" length="4" init="5" />
    <data name="polygon" type="struct" struct="polygon" count="nbrPolygons"
/>
  </recordformat>
</rfml>«
```

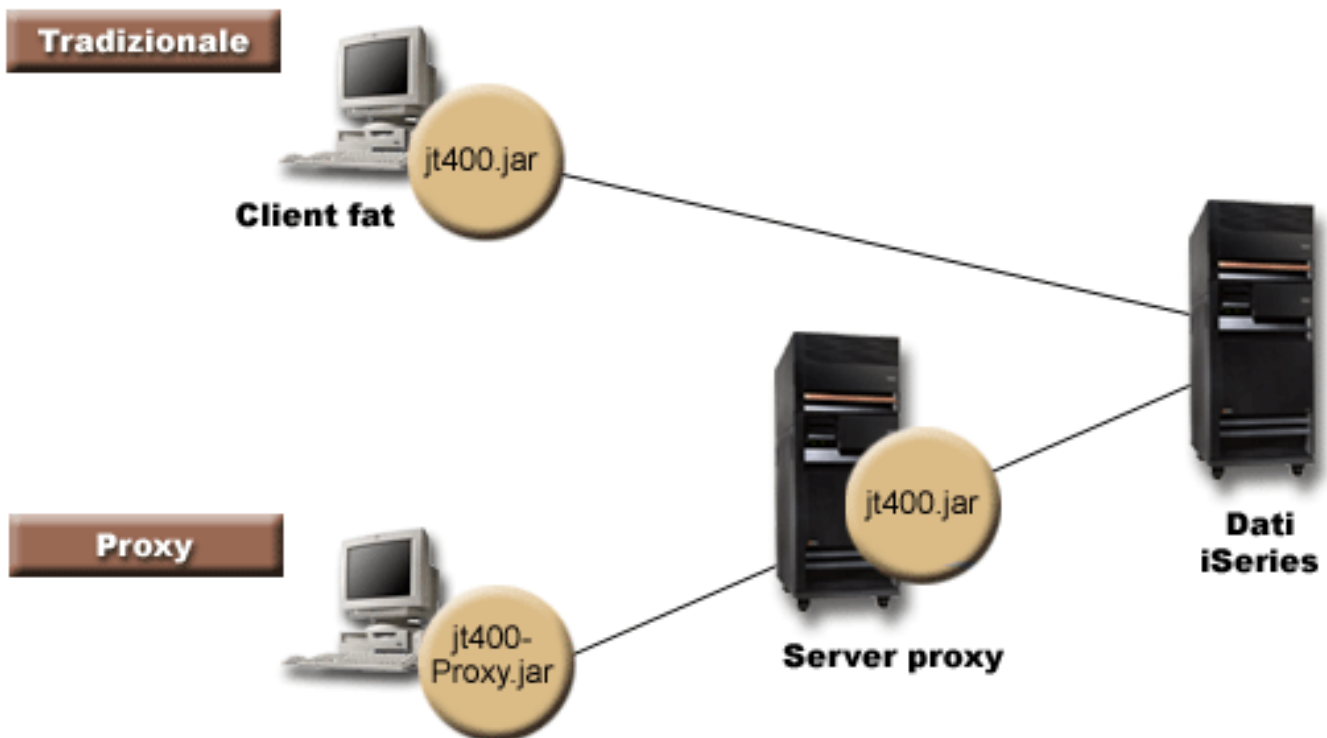
# Supporto proxy

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.

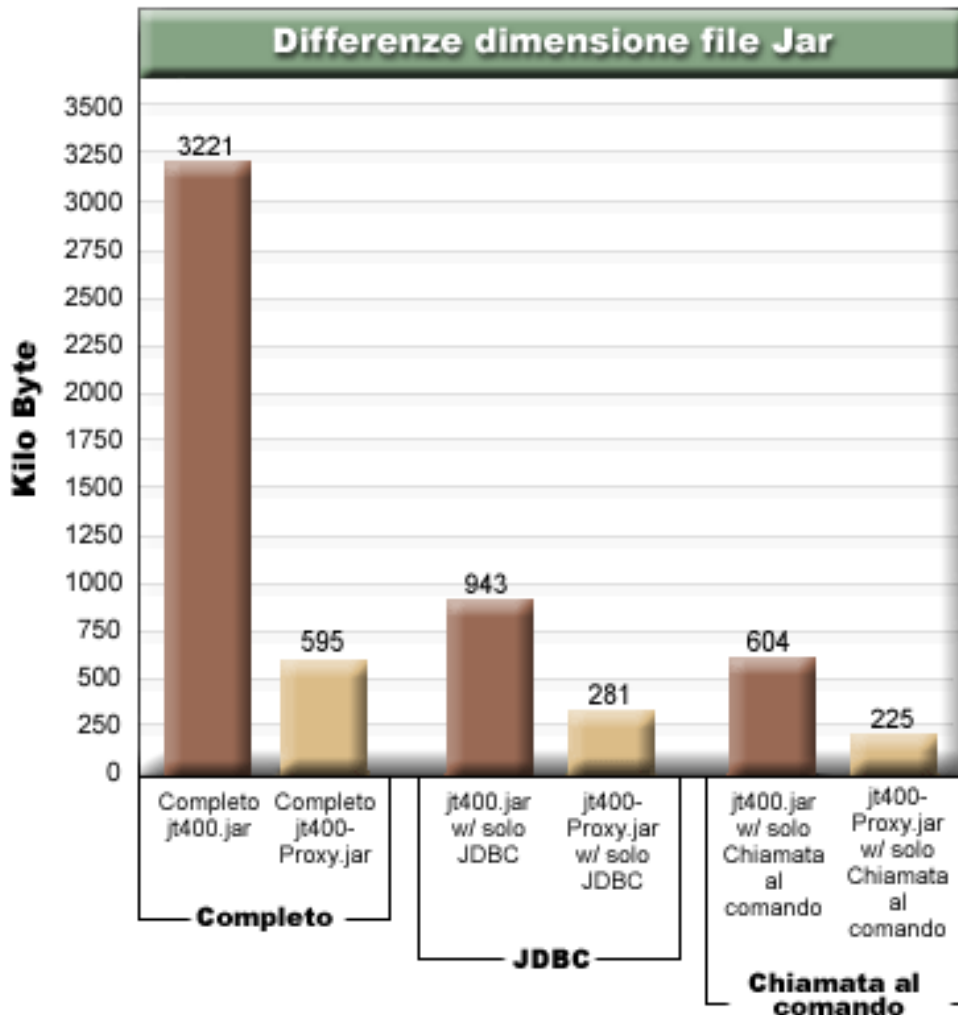
Prima del supporto proxy, le classi contenenti l'interfaccia pubblica, tutte le classi necessarie per l'elaborazione della richiesta e l'applicazione stessa erano eseguite sulla stessa JVM. Quando si utilizza il supporto proxy, è necessario che l'interfaccia pubblica sia associata all'applicazione, ma le classi per l'elaborazione delle richieste possano essere eseguite su una diversa JVM. Il supporto proxy non modifica l'interfaccia pubblica. È possibile eseguire lo stesso programma con la versione proxy dell'IBM Toolbox per Java o con la versione standard.

## Utilizzare il 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 l'apertura di un minor numero di porte nel 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 consiste nel fatto che non sia necessaria l'apertura di un'ulteriore porta nei firewall, dal momento che la comunicazione viene effettuata tramite 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.

## Utilizzare il 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`.

## Utilizzare l'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: utilizzare i server proxy

Vengono forniti tre esempi specifici di utilizzo di un server proxy con le istruzioni precedentemente elencate.

- [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](#)
- [Stampa](#)
- [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.

# »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, denominati file sorgente RFML, rappresentano un'utile sottoserie dei tipi di dati DDS (data description specification) per i file logici e fisici sui sistemi iSeries. E' possibile utilizzare i documenti RFML per gestire le informazioni che si trovano in:

- Record file
- Voci coda dati
- Spazi utente
- Buffer dati arbitrari

**Nota:** per maggiori informazioni riguardo l'utilizzo di DDS per descrivere gli attributi dei dati, consultare il [Riferimento DDS](#).

RFML è molto simile a [PCML \(Program Call Markup Language\)](#), un'altra estensione di XML supportata da 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](#)

Scoprire come l'utilizzo di RFML nella propria applicazione riduce la quantità e a volte la complessità del codice da 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 Toolbox per Java 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 XML Parser. L'XML Parser deve estendere la classe org.apache.xerces.parsers.SAXParser. Toolbox per Java comprende un programma di analisi compatibile, l'XML Parser per Java, che è incluso nel file x4j400.jar Toolbox per Java. Per maggiori informazioni, consultare [File jar](#).

**Nota:** RFML ha gli stessi requisiti del programma di analisi di PCML. Come per PCML, se si preserializza il file RFML, non è necessario includere un XML Parser nel CLASSPATH dell'applicazione per eseguire l'applicazione stessa. ⚡



## » Esempio: utilizzare RFML rispetto ad utilizzare classi Record di Toolbox per Java

Questo esempio illustra le differenze tra l'utilizzo di RFML e delle classi Record di Toolbox per Java.

Utilizzando le classi Record tradizionali, si integrano le specifiche del formato dati con la logica aziendale dell'applicazione. L'aggiunta, la modifica o la cancellazione di un campo indica che è necessario modificare o ricompilare il proprio 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 semplicemente modificare il file RFML, spesso senza dover modificare o ricompilare la propria 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.

La lista riportata sotto illustra come un'applicazione Java potrebbe interpretare un record del cliente utilizzando le classi di Toolbox per Java Record, RecordFormat e FieldDescription:

```
// 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, è qui illustrato come si potrebbe interpretare lo stesso record utilizzando RFML.

Il codice Java utilizzato per interpretare il contenuto del record dati del cliente tramite RFML potrebbe presentarsi nel modo seguente:

```

// Il buffer contiene 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"));

```

## » 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 dovrebbe essere un file di testo denominato qcustcdt.rfml.

**Nota:** leggere l'[Esonero di responsabilità per gli esempi di codice](#) per informazioni legali importanti.

```
<?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="cusrecl">

    <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>

</rfml>
```

```
</recordformat>
```

```
<struct name="balance">
```

```
  <data name="amount" type="zoned" length="6" precision="2" init="7"/>
```

```
</struct>
```

```
</rfml>
```



## » Classe RecordFormatDocument

La [classe RecordFormatDocument](#) consente ai programmi Java di effettuare la conversione tra le rappresentazioni di dati RFML e gli oggetti Record e RecordFormat per l'utilizzo con altri componenti Toolbox for 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.

## Utilizzare la classe RecordFormatDocument con altre classi Toolbox per Java

Utilizzare la classe RecordFormatDocument con le seguenti classi Toolbox per Java:

- Classi orientate a record, comprese le classi file di accesso livello a record (AS400File, SequentialFile e KeyedFile) 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 Toolbox per Java, che leggono e scrivono dati nei formati che RecordFormatDocument non indirizza:

- 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 basa solo sui dati descritti da [DDS \(data description specification\)](#).

## » 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 classi record di Toolbox per Java](#) include un [file sorgente RFML](#) di esempio.

### DTD RFML

La [DTD \(document type definition\) RFM](#) 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

Questo 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  
All rights reserved. Licensed Materials Property of IBM  
US Government Users Restricted Rights  
Use, duplication or disclosure restricted by  
GSA ADP Schedule Contract with IBM Corp.  
-->
```

```
<!-- Convenience entities -->  
<!ENTITY % string "CDATA" <!-- a string of length 0 or  
greater -->  
<!ENTITY % nonNegativeInteger "CDATA" <!-- a non-negative integer -->  
<!ENTITY % binary2 "CDATA" <!-- an integer in range 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)">
```

```
<!-- The document root element -->  
<!ELEMENT rfml (struct | recordformat)+>  
<!ATTLIST rfml  
version %string; #FIXED "4.0"  
ccsid %binary2; #IMPLIED  
>
```

```
<!-- Note: The ccsid is the default value that will be used for any  
contained <data type="char"> elements that do not specify a ccsid. -->
```

```
<!-- Note: RFML does not support nested struct declarations. All struct  
elements are direct children of the root node. -->  
<!ELEMENT struct (data)+>  
<!ATTLIST struct  
name ID #REQUIRED  
>
```

```
<!-- <!ELEMENT recordformat (data | struct)*> -->  
<!ELEMENT recordformat (data)*>
```

```

<!ATTLIST recordformat
      name          ID          #REQUIRED
      description   %string;    #IMPLIED
>
<!-- Note: On the server, the Record "text description" field is limited to
50 bytes. -->

<!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
>
<!-- Note: The 'name' attribute must be unique within a given recordformat.
-->
<!-- Note: On the server, the length of Record field names is limited to 10
bytes. -->
<!-- Note: The 'length' attribute is required, except when type="struct".
-->
<!-- Note: If type="struct", then the 'struct' attribute is required. -->
<!-- Note: The 'ccsid' and 'bidistringtype' attributes are valid only when
type="char". -->
<!-- Note: The 'precision' attribute is valid only for types "int",
"packed", and "zoned". -->

<!-- The standard predefined character entities -->
<!ENTITY quot  "&#34;">      <!-- quotation mark -->
<!ENTITY amp  "&#38;#38;"> <!-- ampersand -->
<!ENTITY apos "&#39;">      <!-- apostrophe -->
<!ENTITY lt   "&#38;#60;"> <!-- less than -->
<!ENTITY gt   "&#62;">      <!-- greater than -->
<!ENTITY nbsp "&#160;">    <!-- non-breaking space -->
<!ENTITY shy  "&#173;">    <!-- soft hyphen (discretionary hyphen) -->
<!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 una lista 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=	<p><i>char</i> Un valore carattere. Un valore dati <i>char</i> viene restituito come un <i>java.lang.String</i>. Per ulteriori informazioni, consultare i <a href="#">valori char per length</a>.</p> <p><i>int</i> Un valore di numero intero. Un valore dati <i>int</i> viene restituito come un <i>java.lang.Long</i>. Per ulteriori informazioni, consultare i <a href="#">valori int per length e precision</a>.</p> <p><i>packed</i> Un valore decimale compresso. Un valore dati <i>packed</i> viene restituito come un <i>java.math.BigDecimal</i>. Per ulteriori informazioni, consultare i <a href="#">valori packed per length e precision</a>.</p> <p><i>zoned</i> Un valore a zonatura decimale. Un valore dati <i>zoned</i> viene restituito come un <i>java.math.BigDecimal</i>. Per ulteriori informazioni, consultare i <a href="#">valori zoned per length e precision</a>.</p> <p><i>float</i> Un valore a virgola mobile. L'attributo <b>length</b> specifica il numero di byte: 4 o 8. Un numero intero a 4 byte viene restituito come un <i>java.lang.Float</i>. Un numero intero a 8 byte viene restituito come un <i>java.lang.Double</i>. Per ulteriori informazioni, consultare i <a href="#">valori float per length</a>.</p> <p><i>byte</i> Un valore byte. Non viene eseguita alcuna</p>	<p>Indica il tipo di dati che si sta utilizzando (char, integer, packed, zoned, floating point, byte o struct).</p> <p>I valori degli attributi length e precision sono differenti per diversi tipi di dati. Per ulteriori informazioni, consultare i valori <a href="#">per length e precision</a>.</p>

	<p>conversione sui dati. Un valore dati <i>byte</i> viene restituito come una schiera di valori <i>byte</i> (<i>byte[]</i>). Per ulteriori informazioni, consultare i <a href="#">valori byte per length</a>.</p> <p><i>struct</i>  Il nome dell'elemento <b>&lt;struct&gt;</b>. Un elemento <i>struct</i> consente di definire una struttura una volta e di riutilizzarla più volte all'interno del documento. Quando si utilizza <b>type="struct"</b>, è come se la struttura specificata venisse visualizzata in questa ubicazione nel documento. Un elemento <i>struct</i> non prevede un valore <i>length</i> e non ha un valore per <i>precision</i>.</p>	
<b>bidstringtype=</b>	<p><i>DEFAULT</i>  dove <i>DEFAULT</i> è il <a href="#">tipo stringa predefinito</a> per i dati non bidirezionali (LTR).</p> <p><i>ST4</i>  dove <i>ST4</i> è <a href="#">Tipo stringa 4</a>.</p> <p><i>ST5</i>  dove <i>ST5</i> è <a href="#">Tipo stringa 5</a>.</p> <p><i>ST6</i>  dove <i>ST6</i> è <a href="#">Tipo stringa 6</a>.</p> <p><i>ST7</i>  dove <i>ST7</i> è <a href="#">Tipo stringa 7</a>.</p> <p><i>ST8</i>  dove <i>ST8</i> è <a href="#">Tipo stringa 8</a>.</p> <p><i>ST9</i>  dove <i>ST9</i> è <a href="#">Tipo stringa 9</a>.</p> <p><i>ST10</i>  dove <i>ST10</i> è <a href="#">Tipo stringa 10</a>.</p> <p><i>ST11</i>  dove <i>ST11</i> è <a href="#">Tipo stringa 11</a>.</p>	<p>Specifica il tipo di stringa bidirezionale per gli elementi <b>&lt;data&gt;</b> con <b>type="char"</b>. 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).</p> <p>I tipi di stringa sono definiti nella <a href="#">Javadoc per la classe BidiStringType</a>.</p>
<b>ccsid=</b>	<p><i>number</i>  dove <i>number</i> definisce un CCSID fisso, immutato.</p> <p><i>data-name</i>  dove <i>data-name</i> definisce il nome che conterrà, nel tempo di esecuzione, il CCSID dei dati carattere. E' possibile che <i>data-name</i> specificato sia un nome completo o un nome relativo all'elemento corrente. I entrambi i casi, è necessario che il nome si riferisca ad un elemento <b>&lt;data&gt;</b> definito con <b>type="int"</b>. Consultare <a href="#">Risolvere i nomi relativi</a> per ulteriori informazioni sulle modalità di risoluzione dei nomi relativi.</p>	<p>Specifica il CCSID host per i dati del carattere per l'elemento <b>&lt;data&gt;</b>. L'attributo <b>ccsid</b> può essere specificato solo per gli elementi <b>&lt;data&gt;</b> con <b>type="char"</b>.</p> <p>Se questo attributo viene omesso, si presume che i dati carattere per questo elemento siano nel CCSID predefinito dell'ambiente host.</p>

<b>count=</b>	<p><i>number</i> dove <i>number</i> definisce un numero di elementi fissi, immutati in una schiera dimensionata.</p> <p><i>data-name</i> dove <i>data-name</i> definisce il nome di un elemento <b>&lt;data&gt;</b> all'interno del documento RFML che conterrà, nel tempo di esecuzione, il numero degli elementi nella schiera. E' possibile che <i>data-name</i> specificato sia un nome completo o un nome relativo all'elemento corrente. In entrambi i casi, è necessario che il nome si riferisca ad un elemento <b>&lt;data&gt;</b> definito con <b>type="int"</b>. Consultare <a href="#">Risolvere i nomi relativi</a> per ulteriori informazioni sulle modalità di risoluzione dei nomi relativi.</p>	<p>Specifica se l'elemento è nella schiera e identifica il numero di voci nella schiera.</p> <p>Se l'attributo <b>count</b> viene omissso, 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.</p>
<b>init=</b>	<p><i>string</i></p>	<p>Specifica un valore iniziale per l'elemento <b>&lt;data&gt;</b>.</p> <p>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.</p>
<b>length=</b>	<p><i>number</i> dove <i>number</i> definisce una lunghezza fissa, immutata.</p> <p><i>data-name</i> dove <i>data-name</i> definisce il nome di un elemento <b>&lt;data&gt;</b> all'interno di un documento RFML che conterrà, nel tempo di esecuzione, la lunghezza. Un <i>data-name</i> può essere specificato solo per gli elementi <b>&lt;data&gt;</b> con <b>type="char"</b> o <b>type="byte"</b>. E' possibile che <i>data-name</i> specificato sia un nome completo o un nome relativo all'elemento corrente. In entrambi i casi, è necessario che il nome si riferisca ad un elemento <b>&lt;data&gt;</b> definito con <b>type="int"</b>. Consultare <a href="#">Risolvere i nomi relativi</a> per ulteriori informazioni sulle modalità di risoluzione dei nomi relativi.</p>	<p>Specifica la lunghezza dell'elemento <b>data</b>. La variazione nell'utilizzo di questo attributo dipende dal tipo di dati. Per ulteriori informazioni, consultare i valori <a href="#">per length e precision</a>.</p>
<b>name=</b>	<p><i>name</i></p>	<p>Specifica il nome dell'elemento <b>&lt;data&gt;</b>.</p>
<b>precision=</b>	<p><i>number</i></p>	<p>Specifica il numero di byte di precisione per alcuni tipi di dati numerici. Per ulteriori informazioni, consultare i valori <a href="#">per length e precision</a>.</p>
<b>struct=</b>	<p><i>name</i></p>	<p>Specifica il nome di un elemento <b>&lt;struct&gt;</b> per l'elemento <b>&lt;data&gt;</b>. Un attributo <b>struct</b> può essere specificato solo per gli elementi <b>&lt;data&gt;</b> con <b>type="struct"</b>.</p>



## » 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
<b>version=</b>	<i>version-string</i> Una versione fissata della <a href="#">DTD RFML</a> . Per V5R2, 4.0 è l'unico valore valido.	Specifica la versione della DTD RFML, che è possibile utilizzare per verificare il valore corretto.
<b>ccsid=</b>	<i>number</i> Un CCSID (coded character set identifier) immutabile.	Specifica il CCSID host, che si applica a tutti gli elementi <b>&lt;data type="char"&gt;</b> contenuti che non specificano un CCSID. Per maggiori informazioni, consultare la tag <a href="#">&lt;data&gt;</a> RFML. Se si omette questo attributo, si utilizza il CCSID predefinito dell'ambiente host.



## » 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	Description
<b>name=</b>	<i>name</i>	Specifica il nome del recordformat.
<b>description=</b>	<i>description</i>	Specifica la descrizione del recordformat.



## » 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	Description
<b>name=</b>	<i>name</i>	Specifica il nome dell'elemento <struct> .





# 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 and 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, 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](#). 

# » Programma di debug del sistema iSeries

Il programma di debug del sistema iSeries IBM fornisce un nuovo ambiente di debug utente grafico sul server iSeries. Utilizzare il programma di debug di sistema iSeries per eseguire il debug e per verificare i programmi che vengono eseguiti sul proprio server iSeries, inclusi quelli eseguiti nell'ambiente OS/400 PASE.

Per ulteriori informazioni sul programma di debug di sistema iSeries, consultare i seguenti argomenti:

## [Componenti](#)

Leggere ciò che riguarda i diversi componenti che comprendono il programma di debug del sistema iSeries e come funzionano assieme per fornire un potente strumento di debug.

## [Installazione](#)

Comprendere ciò che riguarda i requisiti dell'installazione del programma di debug sistema iSeries e imparare ad installarlo sulla stazione di lavoro.

## [Eseguire il programma di debug di sistema iSeries](#)

Vedere come è possibile eseguire i diversi componenti di debug.





# » Componenti del programma di debug del sistema iSeries

Il programma di debug del sistema iSeries è formato dai componenti che seguono:

- Un [Debug Manager](#) basato sul client
- Un [Programma di debug di sistema](#) basato sul client
- Un [Programma di debug OS/400 PASE](#) basato sul cliente
- Un [Hub di debug](#) basato sull'host
- Un [Server di debug](#) basato sull'host

Le descrizioni che seguono forniscono solo informazioni generali relative ai componenti del programma di debug di sistema iSeries. Per ulteriori informazioni relative ai componenti, [eseguire il programma di debug sistema iSeries](#) e consultare l'aiuto in linea. Per visualizzare l'aiuto in linea del programma di debug di sistema iSeries, svolgere una delle seguenti azioni:

- Dal menu **Aiuto** in qualsiasi interfaccia del programma di debug, fare clic su **Aiuto**
- Premere **F1**

## Debug Manager

Debug Manager registra il client con l'[Hub di debug](#), che abilita all'utilizzo della modalità grafica del programma di debug per i sistemi selezionati. Una volta registrato, un client che emette il comando CL Avvio debug (STRDBG) da una sessione di emulazione avvia il programma di debug del sistema.

Utilizzare il Debug Manager per gestire le operazioni e i collegamenti di debug:

- Aggiungere e rimuovere i sistemi
- Aggiungere e rimuovere gli utenti
- Avviare le operazioni di debug
- Avviare il programma di debug di sistema

## Programma di debug di sistema

Utilizzare il programma di debug di sistema per eseguire il debug di programmi che vengono eseguiti sul server iSeries. E' possibile eseguire il debug sui programmi che sono in esecuzione in lavori esistenti sul sistema o utilizzare il programma di debug di sistema per avviare e quindi eseguire il debug sui programmi in un lavoro batch di sistema.

E' possibile impostare il programma di debug di sistema per essere avviato manualmente, automaticamente da una richiesta comandi della stazione di lavoro o tramite l'interfaccia [Debug Manager](#).

Utilizzare il programma di debug di sistema per eseguire le attività di debug che includono:

- Impostare punti di interruzione
- Passare tra i programmi
- collaudare le variabili
- Esaminare lo stack di chiamata
- Esaminare la memoria associata alle variabili di programma
- Esaminare l'attività di sottoprocesso

# Programma di debug OS/400 PASE

Utilizzare il programma di debug OS/400 PASE per eseguire il debug dei programmi che vengono eseguiti in un ambiente OS/400 PASE. E' possibile eseguire il debug dei programmi che sono in esecuzione in un processo esistente sul sistema oppure utilizzare il programma di debug OS/400 PASE per avviare e quindi eseguire il debug di un programma.

Il programma di debug OS/400 PASE può essere avviato direttamente dalla riga comando o utilizzando l'interfaccia [Debug Manager](#).

In aggiunta alle attività di debug elencate precedentemente per il programma di debug di sistema, è possibile utilizzare il programma di debug OS/400 PASE per eseguire le attività di debug specifiche di OS/400 PASE che includono:

- Utilizzare la mappa di caricamento del programma di cui eseguire il debug
- Visualizzare una lista dei file origine e dei metodi
- Tenere traccia dei processi principali e secondari
- Esaminare i registri

## Hub di debug

L'Hub di debug fornisce le funzioni che seguono:

- Opera come un registro per i client che desiderano utilizzare il programma di debug di sistema [»](#) o il programma di debug OS/400 PASE [«](#)
- Gestisce le richieste in entrata per l'avvio dei server di debug

Utilizzare l'interfaccia [Debug Manager](#) per registrare il client con l'Hub di debug. La registrazione di un client memorizza sia le informazioni dell'utente che l'indirizzo TCP/IP del client nel registro. L'utilizzo del comando CL Avvio debug (STRDBG) da una sessione di emulazione fa in modo che l'Hub di debug visualizzi se l'utente che ha eseguito il comando è registrato con Debug Manager. Inoltre verifica se il comando eseguito proviene dallo stesso indirizzo TCP/IP del Debug Manager. Quando questi requisiti vengono soddisfatti, viene avviata l'applicazione grafica del programma di debug di sistema iSeries al posto dell'ambiente di debug tradizionale.

l'Hub di debug serve inoltre come singolo punto di collegamento per tutte le applicazioni del programma di debug di sistema iSeries. Quando un [»](#)componente`«` del programma di debug di sistema iSeries esegue un'operazione di avvio debug, l'hub di debug inoltra un lavoro Server debug per conto degli utenti e lo passa sul collegamento TCP/IP associato.

## Server debug

Il server debug è un server TCP/IP che viene avviato dall'[Hub di debug](#) quando [»](#)uno dei programmi di debug`«` emette una richiesta di avvio del debug. Il lavoro server quindi assiste il lavoro su cui è in esecuzione il debug ed emette i comandi e le API del programma di debug appropriati.`«`

# » Installare il programma di debug di sistema iSeries

Per eseguire il programma di debug di sistema iSeries, la stazione di lavoro client deve soddisfare i requisiti hardware e software che seguono.

## Requisiti hardware

E' necessario che il seguente hardware sia installato sul client:

- CPU: 400 - 500 MHz
- Memoria: minimo 128 MB (si consiglia 256 MB)

## Requisiti software

E' necessario che il seguente software sia installato sul client:

- Uno dei seguenti:
  - Java 2 Platform, sia Standard Edition (J2SE) che Enterprise Edition (J2EE), versione 1.3 o successiva
  - Java 2 Runtime Environment (JRE), Standard Edition, versione 1.3.1 o successiva
- jhall.jar (uno dei file jar in JavaHelp<sup>TM</sup>)

**Nota:** assicurarsi di aggiungere jhall.jar alla variabile di ambiente CLASSPATH del client.

Per ulteriori informazioni relative all'installazione del software menzionato precedentemente, consultare il [sito web](#)

[Sun Java](#) 

## Installare il file jar del programma di debug di sistema iSeries

Il programma di debug di sistema iSeries viene fornito come parte di IBM Toolbox per Java. Se non si ha il file jt400.jar Toolbox per Java già installato sul client, sarà necessario installarlo quando si installa il programma di debug di sistema iSeries.

Prima di installare il programma di debug di sistema iSeries, assicurarsi che il sistema client soddisfi i [requisiti elencati in precedenza](#). Per installare il programma di debug di sistema iSeries, completare le fasi che seguono:

1. [Installare Toolbox per Java](#), assicurarsi di copiare jt400.jar e tes.jar sul client.

**Nota:** se Toolbox per Java è già installato sul server, sia il file jt400.jar che il file tes.jar si trovano sullo stesso indirizzario sul server:

```
/QIBM/ProdData/HTTP/Public/jt400/lib/
```

2. Dopo aver copiato i file jar sul client, aggiungerli alla variabile di ambiente CLASSPATH.

Ora è possibile utilizzare il client per [eseguire il programma di debug di sistema iSeries](#).◀◀

# » Eseguire il programma di debug di sistema iSeries

E' possibile utilizzare una richiesta comandi client per [avviare Debug Manager](#), [avviare il programma di debug di sistema](#) oppure per [avviare il programma di debug OS/400 PASE](#).

Per ulteriori informazioni sul programma di debug di sistema iSeries, avviare il suddetto programma e consultare l'aiuto in linea. Per visualizzare l'aiuto in linea del programma di debug di sistema iSeries, eseguire una delle azioni riportate di seguito:

- Dal menu **Aiuto** in qualsiasi interfaccia del programma di debug, fare clic su **Aiuto**
- Premere **F1**

## Avviare Debug Manager

Per avviare Debug Manager da una richiesta comandi sul client, eseguire il comando:

```
java utilities.DebugMgr
```

## Avviare il programma di debug di sistema

Per avviare il programma di debug di sistema da una richiesta comandi sul proprio client, eseguire il comando riportato di seguito:

```
java utilities.Debug <args>
```

dove <args> rappresenta uno dei seguenti argomenti di comando

- -u = Utente
- -s = Nome sistema
- -j = Descrizione lavoro, nel formato: numero lavoro/utente lavoro/nome lavoro
- -p = Programma da eseguire, nel formato: libreria programma/nome programma

**Nota:** quando si utilizza Debug Manager per registrare il client, è possibile immettere il comando CL Avvio debug (STRDBG) da una sessione di emulazione per avviare il programma di debug di sistema. E' inoltre possibile avviare il programma di debug di sistema direttamente da Debug Manager di sistema.

## Avviare il programma di debug OS/400 PASE

Per avviare il programma di debug OS/400 PASE da una richiesta comandi sul proprio client, eseguire il comando riportato di seguito:


```
java utilities.DebugPASE <args>
```

dove <args> rappresenta uno dei seguenti argomenti di comando

- -u = Utente
- -s = Nome sistema
- -p = Percorso completo del programma da eseguire
- -pid = ID processo

**Nota:** il programma di debug OS/400 PASE può essere avviato direttamente dal Debug Manager di sistema. A differenza del programma di debug di sistema, il programma di debug OS/400 PASE non può essere avviato da una sessione di emulazione.

Per ulteriori informazioni relative al programma di debug di sistema iSeries, eseguire il programma di debug di sistema iSeries e consultare l'aiuto in linea. Per visualizzare l'aiuto in linea del programma di debug di sistema iSeries:

- Dal menu **Aiuto** in qualsiasi interfaccia del Programma di debug di sistema iSeries, fare clic su **Aiuto**
- Da qualsiasi interfaccia del Programma di debug di sistema iSeries, premere **F1** 

## Esempio: file delle proprietà

```
#####  
# IBM Toolbox for Java #  
#-----#  
# Sample properties file #  
# #  
# Name this file jt400.properties and store it in a #  
# com/ibm/as400/access directory that is pointed to by #  
# the classpath. #  
#####  
  
#-----#  
# Proxy server system properties #  
#-----#  
  
# This system property specifies the proxy server host name  
# and port number, specified in the format: hostName:portNumber  
# The port number is optional.  
com.ibm.as400.access.AS400.proxyServer=hqoffice  
  
# This system property specifies which portion of the proxy  
# data flow is encrypted via SSL. Valid values are:  
# 1 - Proxy client to proxy server  
# 2 - Proxy server to AS/400  
# 3 - Proxy client to proxy, and proxy server to AS/400  
com.ibm.as400.access.SecureAS400.proxyEncryptionMode=1  
  
# This system property specifies how often, in seconds,  
# the proxy server will look for idle connections. The  
# proxy server starts a thread to look for clients that are  
# no longer communicating. Use this property to set how  
# often the thread looks for idle connections.  
com.ibm.as400.access.TunnelProxyServer.clientCleanupInterval=7200  
  
# This system property specifies how long, in seconds, a  
# client can be idle before it is cleaned up. The proxy server  
# starts a thread to look for clients that are no longer  
# communicating. Use this property to set long a client can  
# be idle before it is cleaned up.  
com.ibm.as400.access.TunnelProxyServer.clientLifetime=2700  
  
#-----#  
# Trace system properties #  
#-----#  
  
# This system property specifies which trace categories to enable.  
# This is a comma-delimited list containing any combination of trace  
# categories. The complete list of trace categories is defined in  
# the Trace class.  
com.ibm.as400.access.Trace.category=error,warning,information  
  
# This system property specifies the file to which trace output  
# is written. The default is to write trace output to System.out.  
com.ibm.as400.access.Trace.file=c:\\temp\\trace.out
```

```
#-----#
# Command Call system properties                                     #
#-----#

# This system property specifies whether CommandCalls should
# be assumed to be thread-safe. If true, all CommandCalls are
# assumed to be thread-safe. If false, all CommandCalls are
# assumed to be non-thread-safe. This property is ignored
# for a given CommandCall object if either
# CommandCall.setThreadSafe(true/false) or
# AS400.setMustUseSockets(true) has been performed on the object.
com.ibm.as400.access.CommandCall.threadSafe=true
```

```
#-----#
# Program Call system properties                                   #
#-----#

# This system property specifies whether ProgramCalls should
# be assumed to be thread-safe. If true, all ProgramCalls are
# assumed to be thread-safe. If false, all ProgramCalls are
# assumed to be non-thread-safe. This property is ignored
# for a given ProgramCall object if either
# ProgramCall.setThreadSafe(true/false) or
# AS400.setMustUseSockets(true) has been performed on the object.
com.ibm.as400.access.ProgramCall.threadSafe=true
```

```
# End
```

## 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 ()
    {
        /*-----*/
        /* Proxy server system properties */
        /*-----*/

        // 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", "hgoffice");

        // Questa proprietà di sistema specifica quale parte del flusso di
        // dati del
        // proxy è codificato tramite SSL. I valori validi sono:
        // 1 - Client proxy nel server proxy
        // 2 - Server proxy nel server iSeries o AS/400e
        // 3 - Client proxy nel proxy e server proxy nel server iSeries o
        // AS/400e
        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.
// Questa è una lista delimitata da una virgole contenente una
combinazione delle
// categorie di traccia. La lista completa delle categorie di traccia
è definita
// 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
// della traccia. Il valore predefinito consiste nello scrivere
l'emissione della traccia in 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 CommandCalls deve
presupporre
// di essere thread-safe. In caso di true, si presuppone che tutti i
CommandCalls
// siano thread-safe. In caso di false, si presuppone che tutti i
CommandCalls
// vengono considerate non thread-safe. Questa proprietà viene
ignorata
// oggetto CommandCall se
// CommandCall.setThreadSafe(true/false) o
// AS400.setMustUseSockets(true) sono stati eseguiti sull'oggetto.
put ("com.ibm.as400.access.CommandCall.threadSafe", "true");

/*-----*/
/* Program Call system properties */
/*-----*/

// Questa proprietà di sistema specifica se le ProgramCall dovrebbero
// essere considerate thread-safe. Se il valore è true, tutte le
ProgramCall
// vengono considerate thread-safe. Se il valore è false, tutte le
ProgramCall
// vengono considerate non 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");
}
}

```

# »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 ridotta serie di funzioni disponibili nelle classi Access di Toolbox per Java, nel supporto JDBC e altro ancora.


## [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 che aiutano a comprendere come creare ed utilizzare applicazioni senza fili. <<

## » Scaricare ed installare 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 installare 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 [Installare Toolbox per Java sulla stazione di lavoro](#) ed [Installare Toolbox per Java su un server iSeries](#).

# » 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<sup>(TM)</sup> 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 generalmente si trova su un computer differente, di solito 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 e KVM \(K Virtual Machine\)](#) .

## 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 \(KVM\)](#) 
- [MIDP](#) 

## Informazioni collegate

E' possibile utilizzare uno qualsiasi dei numerosi strumenti di sviluppo creati per supportare la creazione di applicazioni Java senza fili. Per una breve lista 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 [pacchetto com.ibm.as400.micro](#) fornisce le classi necessarie per scrivere le applicazioni che consentono all'[unità Tier 0](#) di accedere ai dati e alle risorse del server iSeries.

**Nota:** per utilizzare le classi ToolboxMe per iSeries, è necessario [scaricare ed installare separatamente il componente ToolboxME per iSeries](#).

ToolboxME per iSeries fornisce le seguenti classi:

- [MEServer](#) media le richieste dalla unità Tier0 al server host
- Numerose classi forniscono una sottoserie di funzioni dal pacchetto Access Toolbox per Java
  - [AS400](#) - effettua il collegamento ad un sistema iSeries
  - [CommandCall](#) - effettua una chiamata a un comando iSeries
  - [DataQueue](#) - legge da e scrive su una coda dati del server iSeries
  - [ProgramCall](#) - effettua una chiamata a un programma del server iSeries ed accede ai dati che vengono restituiti dopo aver eseguito il programma
- Altre classi forniscono il [supporto JDBC](#), il quale include anche 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. L'MEServer crea oggetti 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 separatamente ed installare 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. Questa opzione può essere abbreviata utilizzando -po. 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 già 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 ToolboxMe per iSeries AS400 per collegarsi ad un sistema iSeries da una [unità Tier0](#).

**Nota:** per utilizzare le classi ToolboxMe per iSeries, è necessario scaricare separatamente ed installare il componente ToolboxME per iSeries. Per ulteriori informazioni, consultare [Scaricare ed installare TolboxME 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.

Il seguente esempio mostra come utilizzare la classe AS400 per collegarsi ad un sistema 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 installare 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.

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 installare 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 sistema iSeries dove 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.

Il seguente esempio dimostra come utilizzare la classe DataQueue per leggere e scrivere le voci su una coda dati in un sistema 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 separatamente ed installare il componente ToolboxME per iSeries. Per ulteriori informazioni, consultare [Requisiti ed installazione di ToolboxME per iSeries](#).

Per utilizzare il metodo [ProgramCall.run\(\)](#), è necessario fornire i seguenti parametri:

- Il sistema sul quale si desidera eseguire il programma
- Il nome del documento [PCML \(Program Call Markup Language\)](#)
- 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 è semplicemente la comunicazione all'MEServer di quale programma definito da PCML 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 [Program Call Markup Language](#).

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 (MEEException 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](#).

### Accedere e utilizzare i dati

Quando si utilizza un'unità Tier0 per accedere ed aggiornare i dati, si desidera che essa 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 aiutare a 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 sacrificare la possibilità di far diventare 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](#)

### **Conformità SQL**

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. <

## » Utilizzare ToolboxME per iSeries per collegarsi ad un database sul server host

La [classe JdbcMeConnection](#) fornisce una sottoserie di funzioni disponibili nella [classe AS400JDBCConnection](#) di 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 separatamente ed installare il componente ToolboxME per iSeries. Per ulteriori informazioni, consultare [Requisiti ed installazione 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];
```

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 una lista completa delle proprietà supportate da JdbcMeDriver.

### Esempi: utilizzare JdbcMeDriver per il collegamento al server

**Esempio 1:** collegamento al database del server senza specificare uno schema predefinito, una porta o le proprietà JDBC. L'ID utente e la parola d'ordine vengono specificati come parametri nel metodo.

```
// Collegarsi al sistema 'mysystem'. Non vengono specificati
schemi, porte o
// proprietà predefinite.
Connection c = JdbcMeDriver.getConnection
("jdbc:as400://mysystem.helloworld.com;
meserver=myMeServer;
"ouser",
"apassword");
```

**Esempio 2:** collegamento al database del server quando si specificano lo schema e le proprietà JDBC. L'ID utente e la parola d'ordine vengono specificati come parametri nel metodo.

```
// Collegarsi al sistema 'mysystem'. Specificare uno schema e
// due proprietà JDBC. Non specificare una porta.
Connection c2 = JdbcMeDriver.getConnection
("jdbc:as400://mysystem.helloworld.com/mySchema;
meserver=myMeServer;
naming=system;
errors=full;
"ouser",
"apassword");
```

**Esempio 3:** collegamento al database del server; le proprietà (inclusi l'ID utente e la parola d'ordine) vengono specificate utilizzando una URL.

```
        // Collegarsi utilizzando le proprietà. Le proprietà sono 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 4:** scollegarsi dal database. Utilizzare il metodo close() sull'oggetto di collegamento per scollegarsi dal server:

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





## » Classe JdbcMeDriver

La [classe JdbcMeDriver](#) fornisce una sottoserie di funzioni disponibili nella [classe AS400JDBCdriver](#) di 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 separatamente ed installare il componente ToolboxME per iSeries. Per ulteriori informazioni, consultare [Scaricare ed installare ToolboxME per iSeries](#).

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 il seguente codice:

```
Connection c = JdbcMeDriver.getConnection
("jdbc:as400://mysystem.helloworld.com;
 meserver=myMeServer;
 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 gli 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 separatamente ed installare il componente ToolboxME per iSeries. Per ulteriori informazioni, consultare [Scaricare ed installare TolboxME per iSeries](#).

## JdbcMeLiveResultSet

La classe [JdbcMeLiveResultSet](#) fornisce una sottoserie di funzioni disponibili nella [classe AS400JDBCResultSet](#) 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 di risultati per puntare la riga nella serie stessa a cui sta accedendo il programma Java. JDBC 2.0 fornisce metodi aggiuntivi per accedere a specifiche posizioni in un database:

Posizioni del cursore scorrevole	
<a href="#">absolute</a>	<a href="#">moveToInsertRow</a>
<a href="#">first</a>	<a href="#">previous</a>
<a href="#">last</a>	<a href="#">relative</a>
<a href="#">moveToCurrentRow</a>	

## 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 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 insensibile allo scorrimento generalmente 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 IBM Toolbox per Java non supporta serie di risultati insensibili 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 una lista completa dei 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 il tipo di serie di
risultati
// aggiornabile simultaneamente.
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();

```

## Classe JdbcMeOfflineResultSet

La [classe JdbcMeOfflineResultSet](#) fornisce una sottoserie di funzioni disponibili nella [classe AS400JDBCResultSet](#) di 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 potevano 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 Toolbox per Java. Utilizzare JdbcMeResultSetMetaData nell'applicazione client Tier0 per determinare i tipi e le proprietà delle colonne in un JDBCResultSet.

Il seguente esempio mostra come 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 separatamente ed installare il componente ToolboxME per iSeries. Per ulteriori informazioni, consultare [Scaricare ed installare TolboxME 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 l'esempio ToolboxMe per iSeries: [Utilizzare ToolboxME per iSeries, MIDP ed IBM Toolbox per Java](#).

## » Classe JdbcMeStatement

La [classe JdbcMeStatement](#) fornisce una sottoserie di funzioni disponibili nella [classe AS400JDBCStatement](#) di 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 separatamente ed installare il componente ToolboxME per iSeries. Per ulteriori informazioni, consultare [Scaricare ed installare ToolboxME per iSeries](#).

## Classe Istruzioni

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=ouser;
    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();
```



# » Creare ed eseguire un 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](#).

## Creare l'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 una 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 il Palm OS, si dovrebbe creare un file chiamato 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 prescelti 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.

## Eseguire l'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. 



## » Esempi di lavoro di ToolboxME per iSeries

I seguenti esempi di lavoro di ToolboxMe per iSeries illustrano i modi di utilizzare ToolboxME per iSeries con il [MIDP \(Mobile Information Device Profile\)](#). Utilizzare i seguenti collegamenti per visualizzare i file sorgente di esempio selezionati o per scaricare tutti i file exource richiesti per creare le applicazioni senza fili funzionanti di esempio:

[Esempio: ToolboxME per iSeries, MIDP e JDBC](#)

[Esempio: ToolboxME per iSeries, MIDP e IBM Toolbox per Java](#)



[Scaricare gli esempi di lavoro ToolboxME per iSeries](#)

Per ulteriori informazioni su come creare un'applicazione ToolboxME per iSeries, consultare [Creare ed eseguire un programma ToolboxME per iSeries](#).



## 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



# Esempi di codice

La lista seguente fornisce collegamenti ai punti di immissione per molti degli esempi utilizzati in tutto l'argomento IBM Toolbox per Java.

[Classi Access](#)

[Bean](#)

[Graphical Toolbox](#)

[Classi HTML](#)

[PCML](#)

[Classi ReportWriter](#)

[Classi Resource](#)

[RFML](#)

[Classi Security](#)

[Classi Servlet](#)

[Semplici esempi](#)

[Suggerimenti per la programmazione](#)

[ToolboxMe per iSeries](#)

[Classi programma di utilità](#)

[Classi Vaccess](#)

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 che sono forniti in tutta la documentazione delle classi di Access.

## AS400JPing

- [Esempio: utilizzare AS400JPing in un programma Java](#)

## BidiTransform

- [Esempio: utilizzare la classe AS400BidiTransform per convertire testo bidirezionale](#)

## CommandCall

- [Esempio: utilizzare CommandCall per eseguire un comando sul server](#)
- [Esempio: utilizzare CommandCall per richiedere il nome del server, il comando per eseguire e stampare il risultato](#)

## ConnectionPool

- [Esempio: utilizzare AS400ConnectionPool per creare i collegamenti al server](#)

## DataArea

- [Esempio: creare e scrivere in un'area dati decimali](#)

## DataConversion e DataDescription

- [Esempio: utilizzare le classi FieldDescription, RecordFormat e Record](#)
- [Esempio: immettere i dati in una coda](#)
- [Esempio: leggere i dati da una coda](#)

## DataQueue

- [Esempio: come creare un oggetto DataQueue, leggere dati e scollegarsi](#)
- [Esempio: immettere i dati in una coda](#)
- [Esempio: leggere i dati da una coda](#)

## Certificato digitale

- [Esempio: elencare i certificati digitali che appartengono ad un utente](#)

## EnvironmentVariable

- [Esempio: creare, impostare e richiamare le variabili di ambiente](#)

## Eccezioni

- [Esempio: catturare un'eccezione inviata, richiamare un codice di ritorno e visualizzare il testo dell'eccezione](#)

## FTP

- [Esempio: utilizzare la classe FTP per copiare una serie di file da un indirizzario del server](#)
- [Esempio: utilizzare la classe AS400FTP per copiare una serie di file da un indirizzario](#)

## IFS (Integrated file system)

- [Esempi: utilizzare IFSFile](#)
- [Esempio: utilizzare il metodo IFSFile.listFiles\(\) per elencare il contenuto di un indirizzario](#)
- [Esempio: utilizzare le classi IFSFile per copiare i file](#)
- [Esempio: utilizzare le classi IFSFile per elencare il contenuto di un indirizzario](#)
- [Esempio: come utilizzare IFSJavaFile invece di java.io.File](#)
- [Esempio: utilizzare le classi IFSFile per elencare il contenuto di un indirizzario nel server](#)

## JavaApplicationCall

- [Esempio: eseguire un programma nel server dal client che emette "Hello World!"](#)

## JDBC

- [Esempio: utilizzare l'unità di controllo JDBC per creare e popolare una tabella](#)
- [Esempio: utilizzare l'unità di controllo JDBC per interrogare una tabella ed emettere il suo contenuto](#)

## Lavori

- [Esempio: richiamare e modificare le informazioni di lavoro utilizzando la memoria cache](#)
- [Esempio: elencare tutti i lavori attivi](#)
- [Esempio: stampare tutti i messaggi della registrazione lavori per un determinato utente](#)
- [Esempio: elencare le informazioni di identificazione del lavoro per un determinato utente](#)
- [Esempio: richiamare una lista di lavori nel server ed elencare lo stato del lavoro ed il relativo ID](#)
- [Esempio: visualizzare i messaggi nella registrazione lavori per un lavoro che appartiene all'utente corrente](#)

## Coda messaggi

- [Esempio: come utilizzare l'oggetto coda messaggi](#)
- [Esempio: stampare il contenuto di una coda messaggi](#)
- [Esempio: richiamare e stampare un messaggio](#)
- [Esempio: elencare il contenuto della coda messaggi](#)
- [Esempio: utilizzare AS400Message con CommandCall](#)
- [Esempio: utilizzare AS400Message con ProgramCall](#)

## NetServer

- [Esempio: utilizzare un oggetto NetServer per modificare il nome del NetServer](#)

## Stampa

- [Esempio: creare un file di spool da un flusso di immissione](#)
- [Esempio: creare un flusso di dati SCS utilizzando la classe SCS3812Writer](#)
- [Esempio: leggere un file di spool esistente](#)
- [Esempio: creare una lista asincrona di tutti i file di spool utilizzando l'interfaccia PrintObjectListListener](#)
- [Esempio: creare una lista asincrona di tutti i file di spool \*senza\* utilizzare l'interfaccia PrintObjectListListener](#)
- [Esempio: creare una lista asincrona di tutti i file di spool](#)

## Autorizzazione

- [Esempio: impostare l'autorizzazione di un oggetto AS400](#)

## Chiamata al programma

- [Esempio: utilizzare ProgramCall](#)
- [Esempio: utilizzare ProgramCall per richiamare lo stato del sistema](#)
- [Esempio: inoltrare i dati del parametro con un oggetto Program parameter](#)

## QSYSObjectPathName

- [Esempio: creare un nome IFS](#)
- [Esempio: utilizzare QSYSObjectPathName.toPath\(\) per creare un nome oggetto di AS400](#)
- [Esempio: utilizzare QSYSObjectPathName per analizzare il nome del percorso IFS](#)

## Accesso al livello record

- [Esempio: accedere in modo sequenziale ad un file](#)
- [Esempio: utilizzare le classi di accesso al livello record per leggere un file](#)
- [Esempio: utilizzare le classi di accesso al livello record per leggere i record tramite chiave](#)

- [Esempio: utilizzare la classe LineDataRecordWriter](#)

## Chiamata al programma di servizio

- [Esempio: utilizzare ServiceProgramCall per richiamare una procedura](#)

## Stato di sistema

- [Esempio: utilizzare la memorizzazione in cache con la classe SystemStatus](#)

## Lotto di sistema

- [Esempio: impostare la dimensione massima degli errori per SystemPool](#)

## SystemValue

- [Esempio: utilizzare SystemValue e SystemValueList](#)

## Trace

- [Esempio: utilizzare il metodo Trace.setTraceOn\(\)](#)
- [Esempio: miglior utilizzo di Trace](#)
- [Esempio: utilizzare la traccia del componente](#)

## Gruppi di utenti

- [Esempio: richiamare una lista di utenti](#)
- [Esempio: creare una lista di tutti gli utenti di un gruppo](#)

## Spazio utente

- [Esempio: come creare uno spazio utente](#)

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: utilizzare CommandCall

**Nota:** leggere l'[Esonero di responsabilità per gli esempi di codice](#) per informazioni legali importanti.

```
////////////////////////////////////  
//  
// Esempio di chiamata al comando. 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" 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[] parameters)  
    {  
  
        // 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 rappresenta il sistema a cui  
inviare 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: utilizzare AS400ConnectionPool

**Nota:** leggere l'[Esonero di responsabilità per gli esempi di codice](#) per informazioni legali importanti.

```
////////////////////////////////////
//
// Esempio di AS400ConnectionPooling. Questo programma utilizza un
AS400ConnectionPool per
// creare collegamenti ad un sistema 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 di servizio
    // definite nella classe AS400 (FILE, PRINT, COMMAND, DATAQUEUE
ecc.))
    // Poiché i collegamenti sono già stati stabiliti, viene evitato il
tempo normalmente richiesto per il collegamento
    // 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. Questo
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. Questo 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 since
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();
    }
}
}
}

```

## Esempio: utilizzare le 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 di Data Queue. Questo programma utilizza la classe DataQueue per
// inserire
// 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
// ed i numeri sono convertiti da Java al formato server. Dal momento in
// cui 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.
//
// Questo rappresenta il lato produttore 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 questo programma è stato eseguito. Viene catturata ed
ignorata l'eccezione La coda
// esiste già.
        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 è quanto
        // viene effettivamente 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: utilizzare le 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 di Data Queue. Questo programma utilizza le classi Data Queue
per leggere
// voci da un coda dati sul server. Le voci sono state inserite nella
// coda tramite il programma di esempio DQProducer.
//
// Questo 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);
}
}

```

# Esempi: utilizzare IFSFile

Gli esempi che seguono mostrano come utilizzare la classe IFSFile:

- **Esempio:** [creare un indirizzario](#)
- **Esempio:** [utilizzare eccezioni IFSFile per la traccia degli errori](#)
- **Esempio:** [elencare i file con un'estensione .txt](#)
- **Esempio:** [utilizzare il metodo IFSFile listFiles\(\) per elencare il contenuto di un indirizzario](#)

## Esempio: creare 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: utilizzare le 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 ");
}

// La cancellazione ha dato esito negativo.
Richiamare il codice // 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: elencare i file con un'estensione .txt

Esempio 3: 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, "/");

// Generare una lista 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: utilizzare il 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 integrated
file system
// 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);

                // Generare una lista di IFSFiles. Passare il metodo
listFiles,
```

```

        // l'oggetto filtro indirizzario ed i criteri di ricerca
        // corrispondenti. Questo 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.
        //
informazioni // Tuttavia, con l'utente del metodo listFiles, le
dal // sull'attributo non verranno aggiornate automaticamente
potrebbero diventare // server. Ciò significa che le informazioni sull'attributo
        // incoerenti rispetto alle 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 sulla lista.
            // 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 riuscita la lista
        // 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
// in base alle informazioni nell'oggetto file. Questo consente alla
funzione
// mainline di decidere cosa fare riguardo alla lista 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
            // di restituire questo file nella lista di voci restituita nel
            // metodo .list().
            return true;
        }

        catch (Exception e)
        {
            return false;
        }
    }
}
```

## Esempio: utilizzare le 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 di 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: utilizzare le 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 integrated file
system
// 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);

                // Generare la lista di nomi. Passare il metodo della lista,
                // l'oggetto filtro indirizzario ed i criteri di ricerca
            }
        }
    }
}
```

```

corrispondenti.
    //
    // Nota - questo esempio esegue l'elaborazione nell'oggetto
    // filtro. Un'alternativa è rappresentata dall'elaborare la
lista dopo
    // che è stata restituita dalla chiamata al metodo della lista.

    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 riuscita la lista
        // 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  
// in base alle informazioni nell'oggetto file. Questo consente alla  
funzione  
// mainline di decidere cosa fare riguardo alla lista 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 nella lista di voci restituita nel  
            // metodo .list().
```



```
        return true;
    }

    catch (Exception e)
    {
        return false;
    }
}
```

## Esempio: utilizzare JDBCPopulate per creare e popolare una tabella

**Nota:** leggere l'[Esonero di responsabilità per gli esempi di codice](#) per informazioni legali importanti.

```
////////////////////////////////////  
//  
// Esempio di JDBCPopulate. Questo programma utilizza il programma 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());

        // Stabilire un collegamento al database. Poichè non vengono
forniti
        // 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
        // si esegue più volte, è più opportuno utilizzare una
        // PreparedStatement e contrassegnare 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: utilizzare JDBCQuery per interrogare una tabella

**Nota:** leggere l'[Esonero di responsabilità per gli esempi di codice](#) per informazioni legali importanti.

```
////////////////////////////////////
//
// Esempio di 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());

        // Stabilire un collegamento al database. Poiché non vengono
forniti
        // 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) {
columnWidths[i-1]));
            System.out.print (format (rsmd.getColumnLabel(i),
                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.isNull ())
            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: utilizzare 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 di identificazione del lavoro relative
// ad 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 a 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 la lista di lavori attivi sul server.
        JobList jobList = new JobList(as400);

        // Richiamare la lista dei lavori attivi.
        Enumeration list = jobList.getJobs();

        // Per ogni lavoro nella lista ...
        while (list.hasMoreElements())
        {
            // Estrarre un lavoro dalla lista.  Se è stato
specificato   // un ID utente stampare le informazioni di
            // solo se l'utente del lavoro corrisponde all'ID
identificazione // non è stato specificato alcun ID utente stampare le
utente. Se      // relative ad ogni lavoro sul sistema.
            // informazioni
            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: utilizzare JobList per richiamare una lista 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" nell'IBM
// Toolbox per Java.  Richiama una lista 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 ed il terzo sono
        facoltativi.  Essi
        // sono l'id utente e la parola d'ordine.  Convertire id utente e
        parola d'ordine
        // in lettere maiuscole 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 lista lavori. Il parametro di immissione è
l'AS400 da quale si desiderano
        // le informazioni sul lavoro.
        JobList jobList = new JobList(as400);

        // Richiamare una lista dei lavori in esecuzione sul server.
        Enumeration listOfJobs = jobList.getJobs();

        // Per ogni lavoro presente nella lista 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 della
lista. 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
        // di emissione. Si assegnerà un 1k di buffer 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".
        // Nome deve contenere 10 caratteri, utente deve essere composto 10
caratteri e
        // numero da 6 caratteri. Si utilizzerà 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. Rimarrà 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

```

```

    {
        // altrimenti 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: utilizzare 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.  Esso visualizzerà i messaggi nella registrazione  
lavori  
// relativi ad un 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 stato passato  
            // come primo argomento della riga comandi.  Se un id utente  
            // ed una 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 lista lavori. Questo oggetto verrà
utilizzato per richiamare
        // la lista dei lavori attivi sul sistema. Una volta recuperata
        // la lista di lavori, il programma individuerà il lavoro
relativo
        // all'utente corrente.
        JobList jobList = new JobList(system);

        // Richiamare la lista di lavori attivi sull'AS/400
Enumeration list = jobList.getJobs();

        boolean Continue = true;

        // Esaminare la lista 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 individuato 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());

                // 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 individuato 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: creare 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 sistema 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);  
            }  
        } 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: creare i 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.

**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)                // Impostare l'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 occurred while creating spooled
file. " + e);
        System.exit(0);
    }
}
}
```

## Esempio: leggere i 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. I byte effettivamente letti verranno restituiti.
            // 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: elencare i 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 illustra l'elencazione di tutti i file di spool su un sistema
// in modo Asincrono utilizzando
// l'interfaccia PrintObjectListListener per ottenere feedback mentre viene
// creata la lista.
// L'elencazione in modo Asincrono consente al chiamante di avviare
// l'elaborazione della lista di oggetti
// prima che l'intera lista sia creata 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 sistema 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 la lista, viene immediatamente restituito
openAsynchronously
    splfList.openAsynchronously();

    do
    {
        // attendere che la lista contenga almeno 25 oggetti o che
venga eseguita
        waitForWakeUp();

        fCompleted = splfList.isCompleted();
        size = splfList.size();

        // emettere i nomi di tutti gli oggetti aggiunti alla lista
        // 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 la lista

```



```

        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 la lista viene aggiornata o
termina.
private synchronized void waitForWakeUp()
    throws InterruptedException
{
    // non disattivarsi se il listener informa che la lista è stata
    eseguita
    if (!fListCompleted)
    {
        wait();
    }
}

// I seguenti metodi implementano l'interfaccia PrintObjectListListener

// Questo metodo è richiamato quando si chiude la lista.
public void listClosed(PrintObjectListEvent event)
{
    System.out.println("*****The list was closed*****");
    fListClosed = true;
    synchronized(this)
    {
        // Impostare l'indicatore in modo da indicare che la lista è
        stata
        // completata ed attivare il sottoprocesso di foreground.
        fListCompleted = true;
        notifyAll();
    }
}

// Questo metodo viene richiamato quando la lista è stata completata.
public void listCompleted(PrintObjectListEvent event)
{
    System.out.println("*****The list has completed*****");
    synchronized (this)
    {
        // Impostare l'indicatore in modo da indicare che la lista è
        stata
        // completata ed attivare il sottoprocesso di foreground.
        fListCompleted = true;
        notifyAll();
    }
}

```

```

    }
}

// Questo metodo viene richiamato se si verifica un errore durante il
recupero
// della lista.
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 la lista è
stata
        // completata ed attivare il sottoprocesso di foreground.
        fListCompleted = true;
        notifyAll();
    }
}

// Questo metodo viene richiamato quando si apre la lista.
public void listOpened(PrintObjectListEvent event)
{
    System.out.println("*****The list was opened*****");
    listObjectCount = 0;
}

// Questo metodo viene richiamato quando un oggetto viene aggiunto alla
lista.
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: elencare i 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 l'apertura della
lista il chiamante
// può svolgere del lavoro supplementare prima di attendere il completamento
della lista.
//
////////////////////////////////////
//
// 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 lista, 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 della lista....

    System.out.println(" Now wait for list to complete.");

    // attendere il completamento della lista
    splfList.waitForListToComplete();

    Enumeration enum = splfList.getObjects();

    // emettere il nome di tutti gli oggetti nella lista
    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 la lista
    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: elencare i 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'elencazione di tutti i file di spool su un sistema  
// in modo Sincrono.  
// L'elencazione in modo Sincrono non ritorna al chiamante fino a quando non  
// viene creata la lista  
// completa. L'utente percepisce un tempo di risposta più lento rispetto  
// all'elencazione in modo Asincrono.  
//  
////////////////////////////////////  
//  
// 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");  
  
            // aperta lista, openSynchronously() restituito al completamento  
            della lista.  
            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 la lista
        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: utilizzare ProgramCall

**Nota:** leggere l'[Esonero di responsabilità per gli esempi di codice](#) per informazioni legali importanti.

```
////////////////////////////////////  
//  
// Esempio di chiamata al programma. Questo programma chiama il programma  
// del 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  
                // programma. Supporre che il primo parametro sia il nome del  
                sistema.  
  
                AS400 as400 = new AS400(parameters[0]);  
  
                // Creare il percorso al programma.  
  
                QSYSObjectPathName programName = new QSYSObjectPathName("QSYS",  
"QWCRSSTS",  
"PGM");
```



```

// Creare l'oggetto chiamata al programma. Associare tale
oggetto all'oggetto
// AS400 che rappresenta il server da cui si richiama lo stato.

ProgramCall getSystemStatus = new ProgramCall(as400);

// Creare la lista parametri del programma. Questo programma
dispone di cinque
// parametri che verranno aggiunti alla lista.

ProgramParameter[] parmlist = new ProgramParameter[5];

// Il programma del 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 );

// Parametro 2 è la dimensione buffer del parametro 1. Si
tratta di un parametro di immissione
// numerico. Impostare il relativo valore su 64, convertirlo al
formato server,
// quindi aggiungere il parametro alla lista di parametri.

AS400Bin4 bin4 = new AS400Bin4( );
Integer iStatusLength = new Integer( 64 );
byte[] statusLength = bin4.toBytes( iStatusLength );
parmlist[1] = new ProgramParameter( statusLength );

// Parametro 3 è il parametro del formato-stato. Si tratta di un
parametro di immissione
// di stringa. Impostare il valore di stringa, convertirlo nel
formato server,
// quindi aggiungere il parametro alla lista di parametri.

AS400Text text1 = new AS400Text(8, as400);
byte[] statusFormat = text1.toBytes("SSTS0200");
parmlist[2] = new ProgramParameter( statusFormat );

// Parametro 4 è il parametro statistiche-reimpostazione. Si
tratta di un parametro di immissione
// di stringa. Impostare il valore di stringa, convertirlo nel
formato server,
// quindi aggiungere il parametro alla lista di parametri.

AS400Text text3 = new AS400Text(10, as400);
byte[] resetStats = text3.toBytes("*NO      ");
parmlist[3] = new ProgramParameter( resetStats );

```

```

        // Parametro 5 è il parametro delle informazioni sull'errore.
Si tratta di un parametro di
        // Immissione/Emissione. Aggiungerlo alla lista dei parametri.

        byte[] errorInfo = new byte[32];
        parmlist[4] = new ProgramParameter( errorInfo, 0 );

        // Impostare il programma da chiamare e la lista di parametri
sull'oggetto
        // chiamata al programma.

        getSystemStatus.setProgram(programName.getPath(), parmlist );

        // Eseguire il programma quindi rimanere inattivi. Si esegue il
programma due volte poiché
        // 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 la lista
dei 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
sono 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 questo 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 questo 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
        // server dei comandi 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: utilizzare le classi di accesso al livello record

**Nota:** leggere l'[Esonero di responsabilità per gli esempi di codice](#) per informazioni legali importanti.

```
////////////////////////////////////  
//  
// Esempio di accesso al livello record. Questo programma richiederà  
all'utente  
// il nome del server ed il file da visualizzare. Il file deve esistere  
// e dovrebbe contenere dei record. Ogni record nel file verrà visualizzato  
// in System.out.  
//  
// Sintassi di chiamata: java RLSequentialAccessExample  
//  
// Questo sorgente è un esempio di "RecordLevelAccess" di IBM Toolbox per  
Java  
//  
////////////////////////////////////  
  
import java.io.*;  
import java.util.*;  
import com.ibm.as400.access.*;  
  
public class RLSequentialAccessExample  
{  
    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 = "";  
        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: utilizzare le 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 informazioni legali importanti.

```
////////////////////////////////////  
//  
// Esempio di accesso a 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 si dovrebbe fare una copia di QCUSTCDT ed aggiornare la copia.  
//  
// Questo sorgente è un esempio di "Accesso al 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");

        // Dovrebbe essere specificato 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 l'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 codice seguente estrae campi
// del record in base la 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: utilizzare 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 informazioni legali importanti.

```
////////////////////////////////////  
//  
// Esempio di accesso a 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 al 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 sar  uguale al formato record per il file QIWS/QCUSTCDT.
    Tuttavia si render 
            // il campo CUSNUM un campo chiave.
            AS400FileRecordDescription recordDescription = new
    AS400FileRecordDescription(system, "/QSYS.LIB/QIWS.LIB/QCUSTCDT.FILE");

            // Vi   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 verr  creato sul server. Si utilizza un oggetto
    QSYSObjectPathName
        // 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 l'accesso di sola lettura. Poiché si accederà
        // casualmente al file, specificare un fattore di blocco di 1. Il
        // 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 codice seguente estrae campi
                // del record in base la 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: utilizzare UserList per elencare tutti gli utenti in un determinato gruppo

**Nota:** leggere l'[Esonero di responsabilità per gli esempi di codice](#) per informazioni legali importanti.

```
////////////////////////////////////  
//  
// 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 lista utenti.  
            UserList userList = new UserList (system);  
  
            // Richiamare una lista degli utenti in un dato gruppo.  
            userList.setUserInfo (UserList.MEMBER);  
            userList.setGroupInfo (groupName);  
            Enumeration enum = userList.getUsers ();  
  
            // Ripetere nella lista 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 tutta la documentazione degli argomenti relativi ai bean.

- [Esempio: utilizzare i listener per stampare un commento quando ci si collega e scollega dal sistema e si eseguono comandi](#)
- [Esempio: utilizzare applet e 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**

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: 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 Beans. 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 commandCall
        // chiamerà 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);
}
}

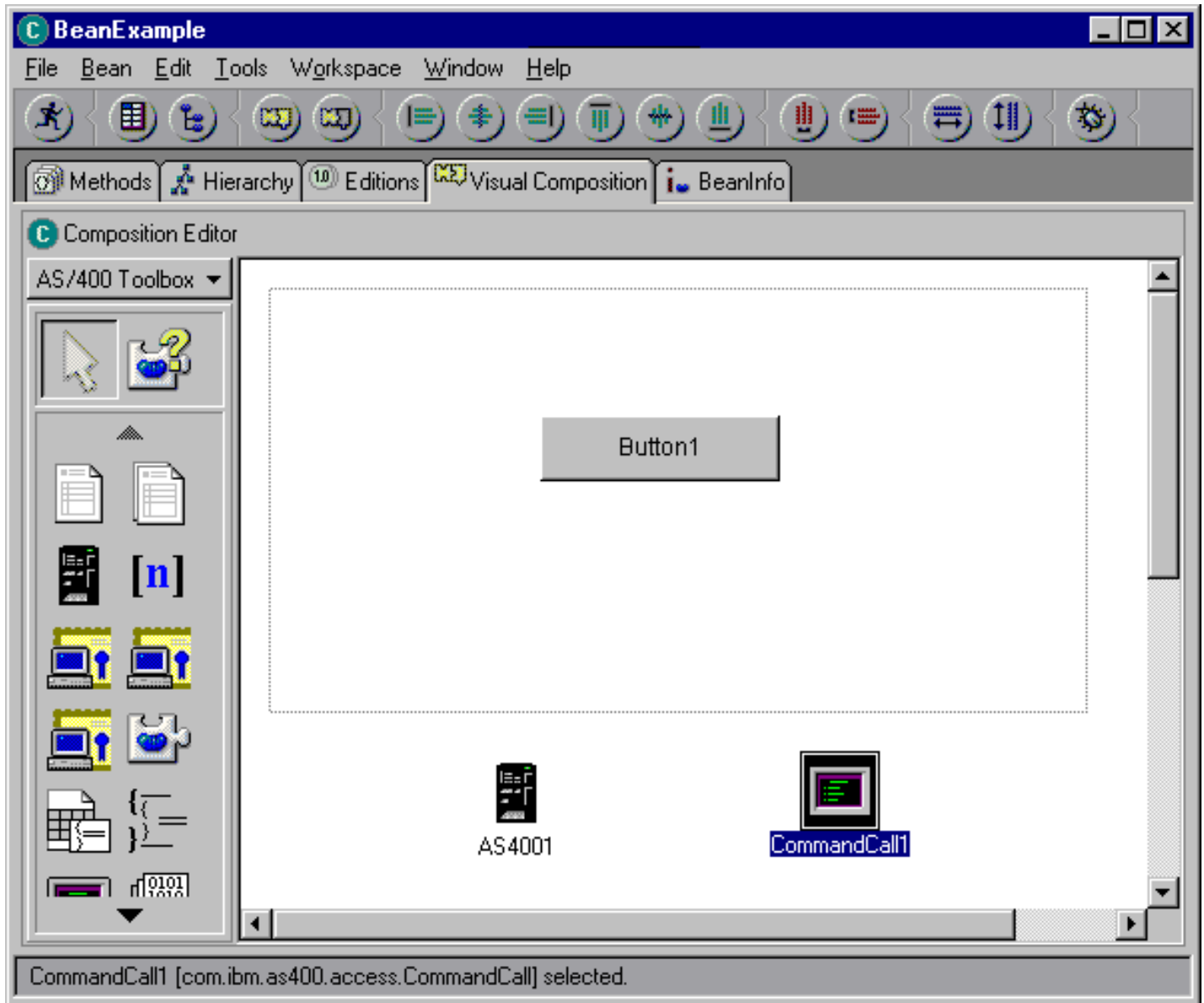
```

# Esempio di codice del programma di creazione bean del visuale

Questo esempio utilizza IBM VisualAge per Java Enterprise Edizione V2.0 Composition Editor, ma altri programmi di creazione di bean visuale sono simili. Questo esempio crea un'applet per un pulsante che, quando viene premuto, esegue un comando sul server iSeries o AS/400e.

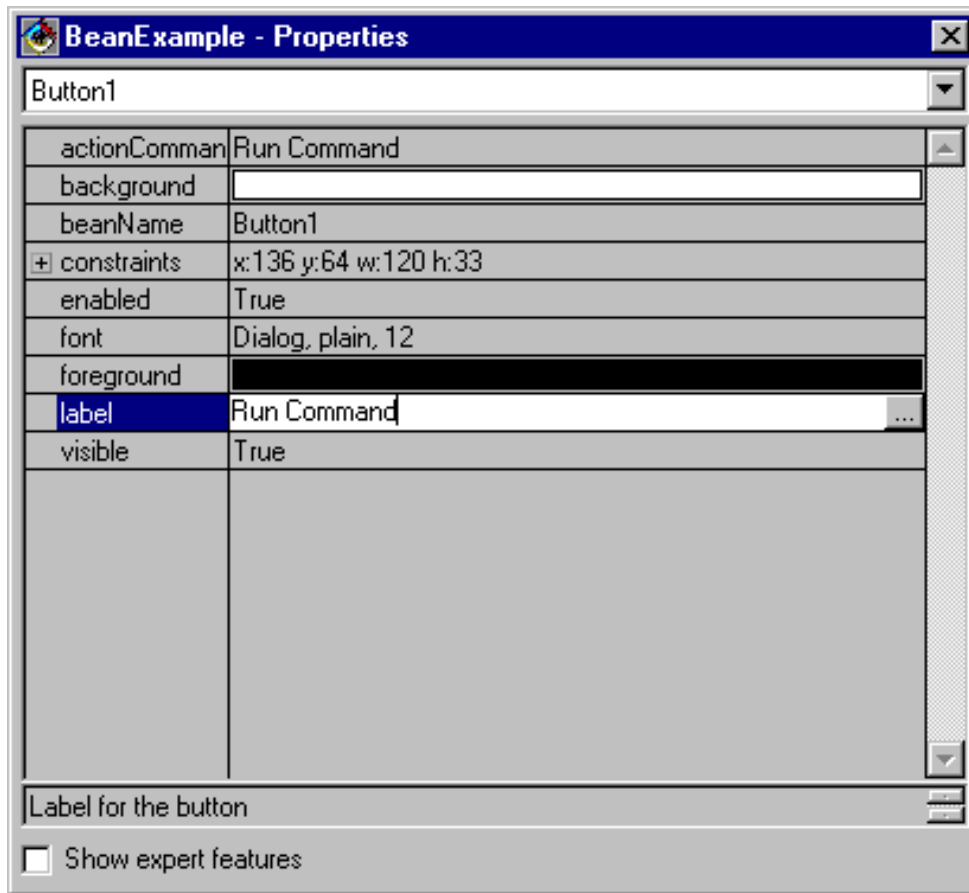
- Trascinare e rilasciare un pulsante sull'applet. (Il pulsante può essere trovato nel programma di creazione bean sulla parte sinistra del separatore della Composizione visuale nella Figura 1.)
- Rilasciare un bean CommandCall e un bean AS400 al di fuori dell'applet. (I bean si trovano nel programma di creazione di bean nella parte sinistra del separatore Composizioni visuale nella Figura 1.)

Figura 1: finestra Editor di Composizione visuale VisualAge - gui.BeanExample



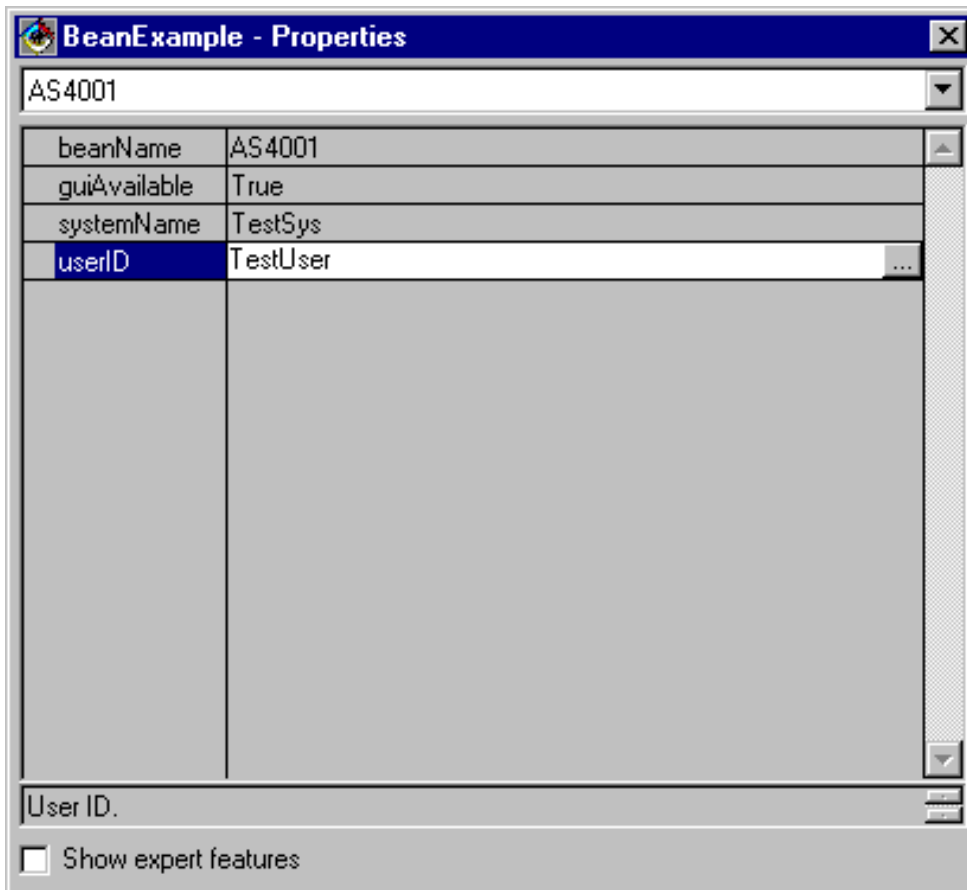
- Modificare le proprietà del bean. (Per effettuare la modifica, selezionare il bean e quindi fare clic con il tasto destro del mouse per visualizzare una finestra a comparsa, che ha l'opzione Proprietà.)
  - Modificare l'etichetta del pulsante in **Esegui comando**, come mostrato nella Figura 2.

Figura 2: modifica dell'etichetta del pulsante in Esegui comando



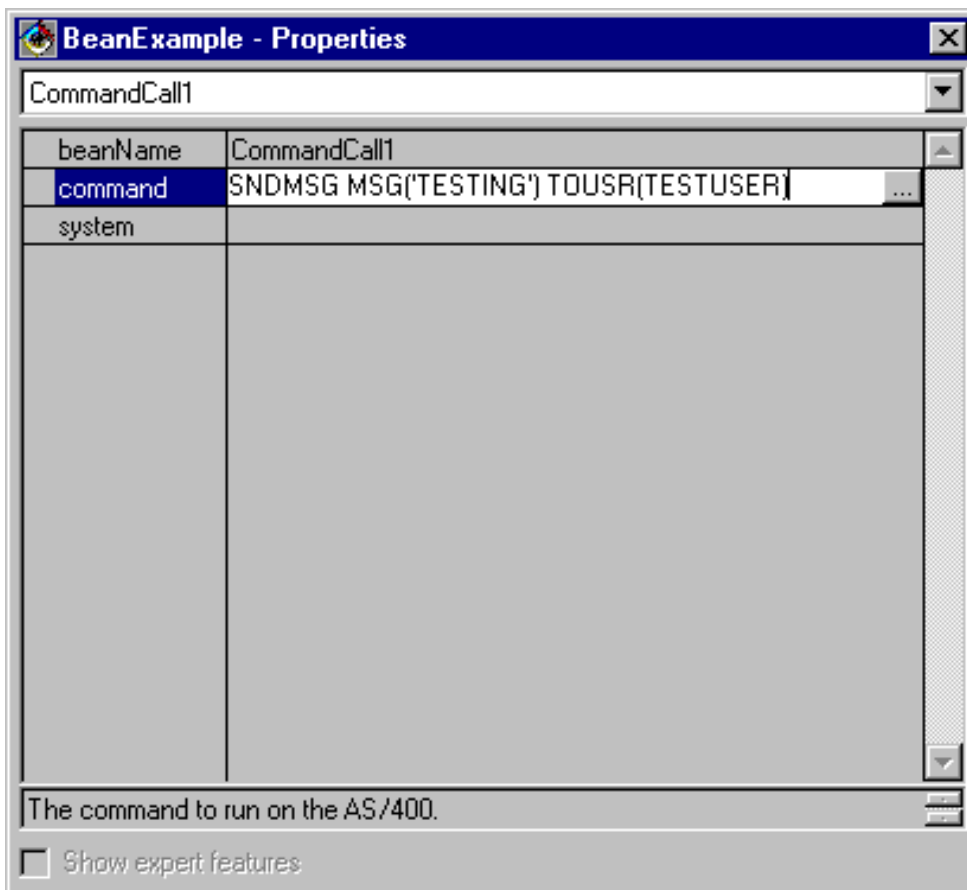
- Modificare il nome del sistema del bean AS400 in **TestSys**
- Modificare l'ID utente del bean AS400 in **TestUser**, come mostrato nella Figura 3.

**Figura 3: modifica del nome dell'ID utente in TestUser**



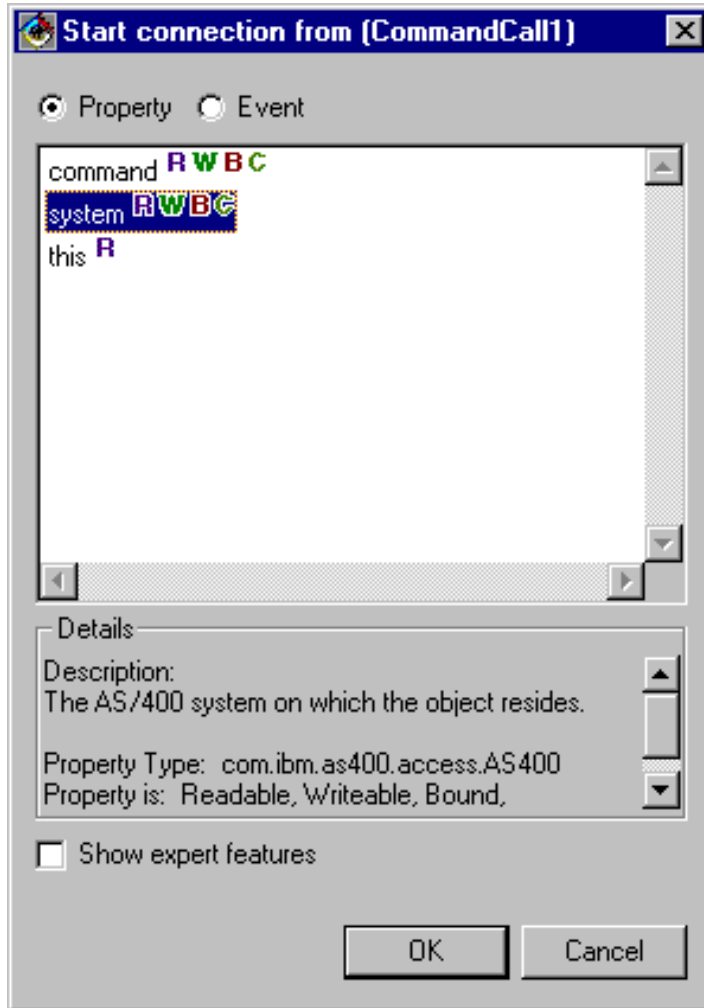
- Modificare il comando del bean CommandCall in `SNDMSG MSG('Testing') TOUSR('TESTUSER')`, come mostrato in Figura 4.

**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** dalla lista di dispositivi come mostrato nella Figura 5.
  - Selezionare il bean AS400
  - Selezionare **questo bean** dal menu a comparsa visualizzato sul bean AS400

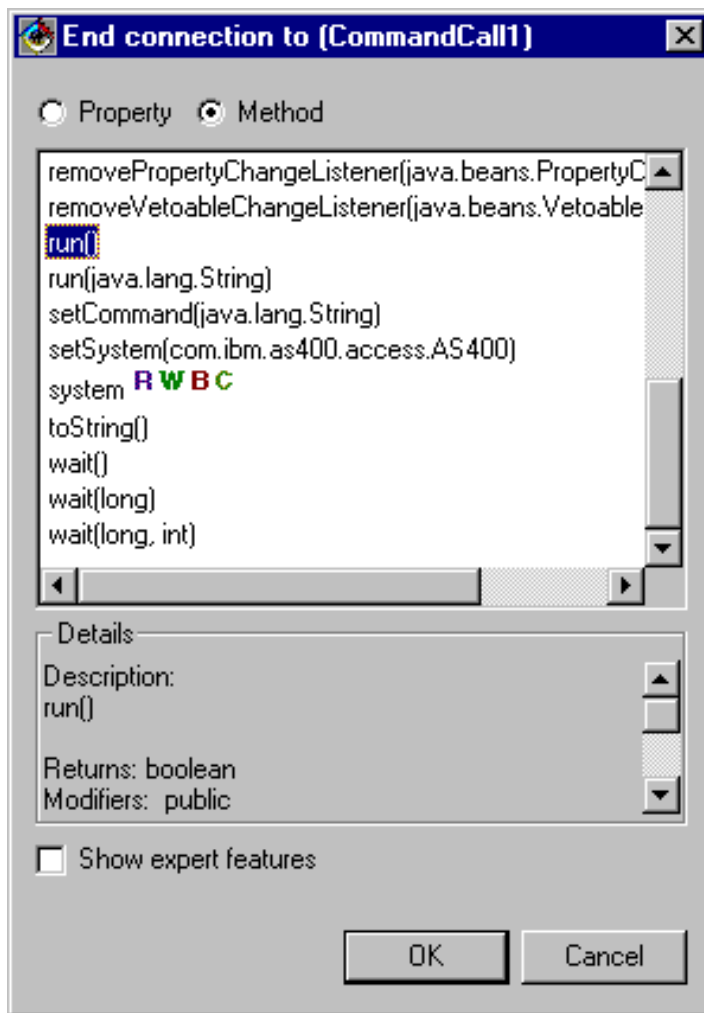
**Figura 5: collegamento del bean AS400 al bean CommandCall**



- 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()** dalla lista dei metodi come mostrato in 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 VisualAge - Esempio di bean terminato**

BeanExample

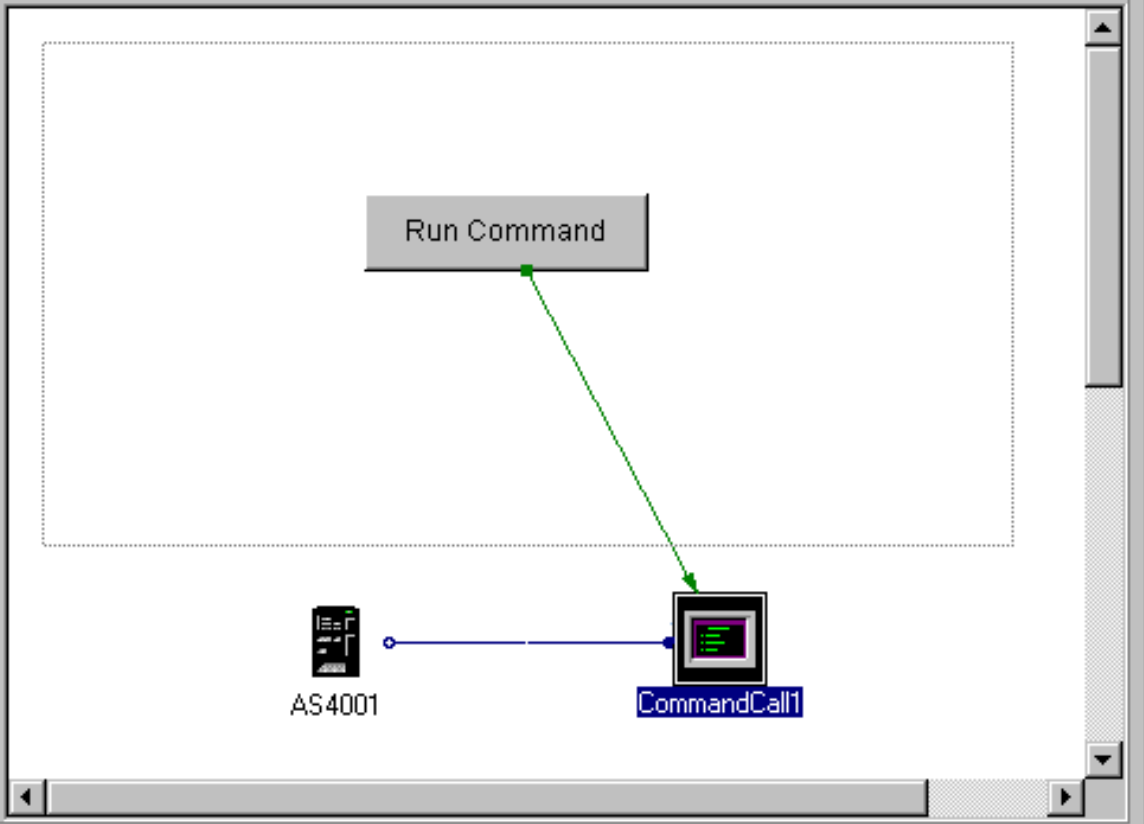
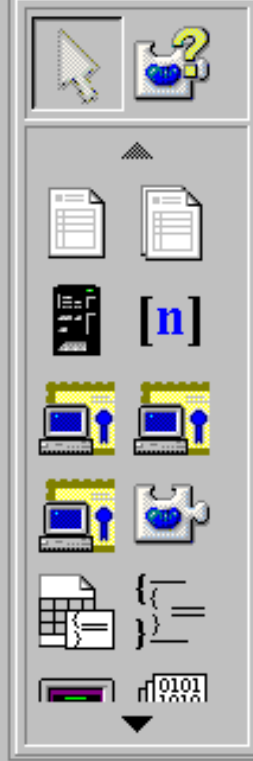
File Bean Edit Tools Workspace Window Help



Methods Hierarchy Editions Visual Composition BeanInfo

Composition Editor

AS/400 Toolbox



CommandCall1 [com.ibm.as400.access.CommandCall] selected.

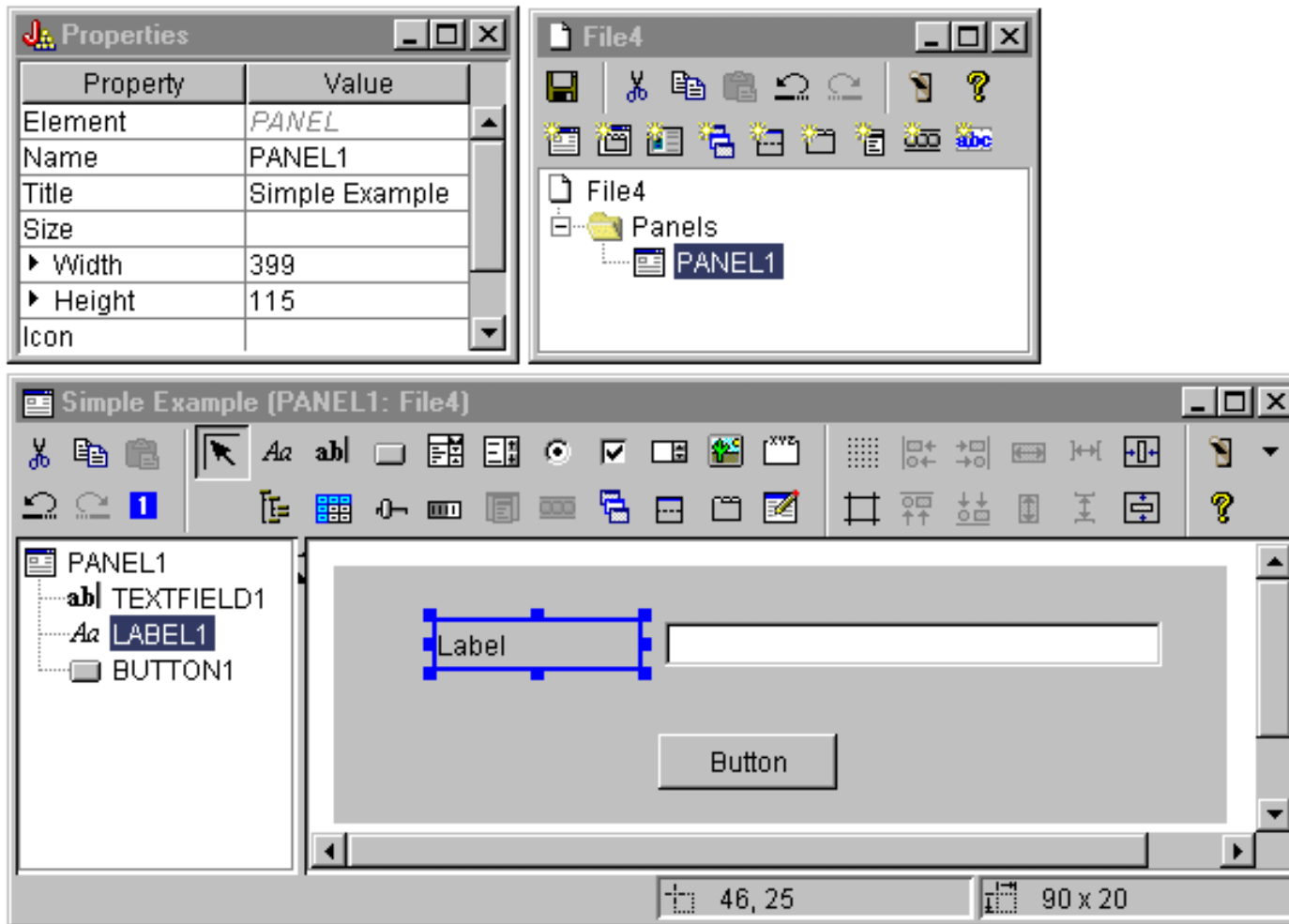
# Esempio: creare un pannello con il GUI Builder

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.

## Costruire il pannello

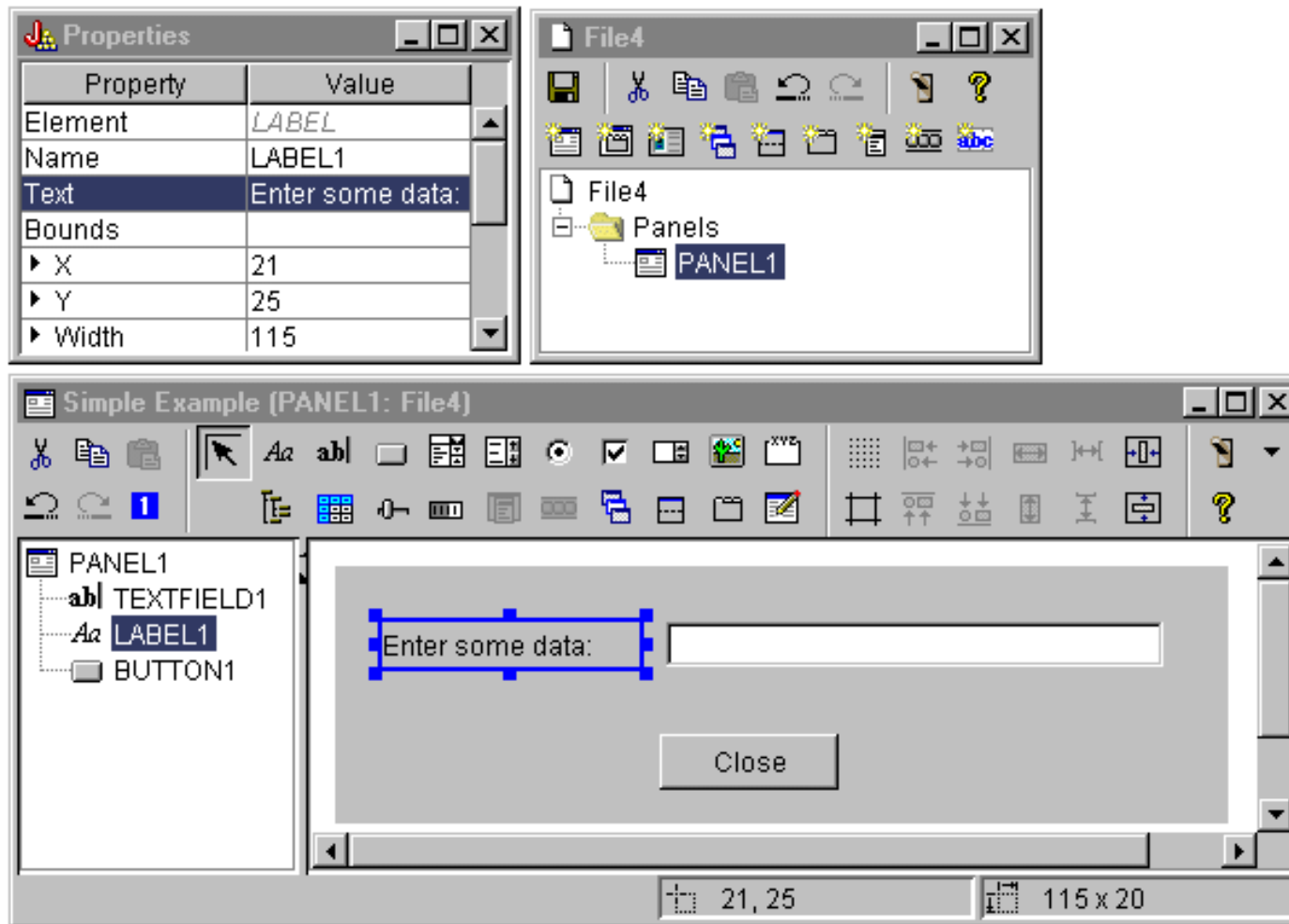
Quando si avvia il GUI Builder, vengono visualizzate le finestre Proprietà e GUI Builder. Creare un nuovo file denominato "MyGUI.pdml". Per questo esempio, inserire un nuovo pannello. Fare clic sull'icona "Inserisci pannello" nella finestra Builder del file. Il suo nome è "PANEL1". Modificare il titolo 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 in 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".

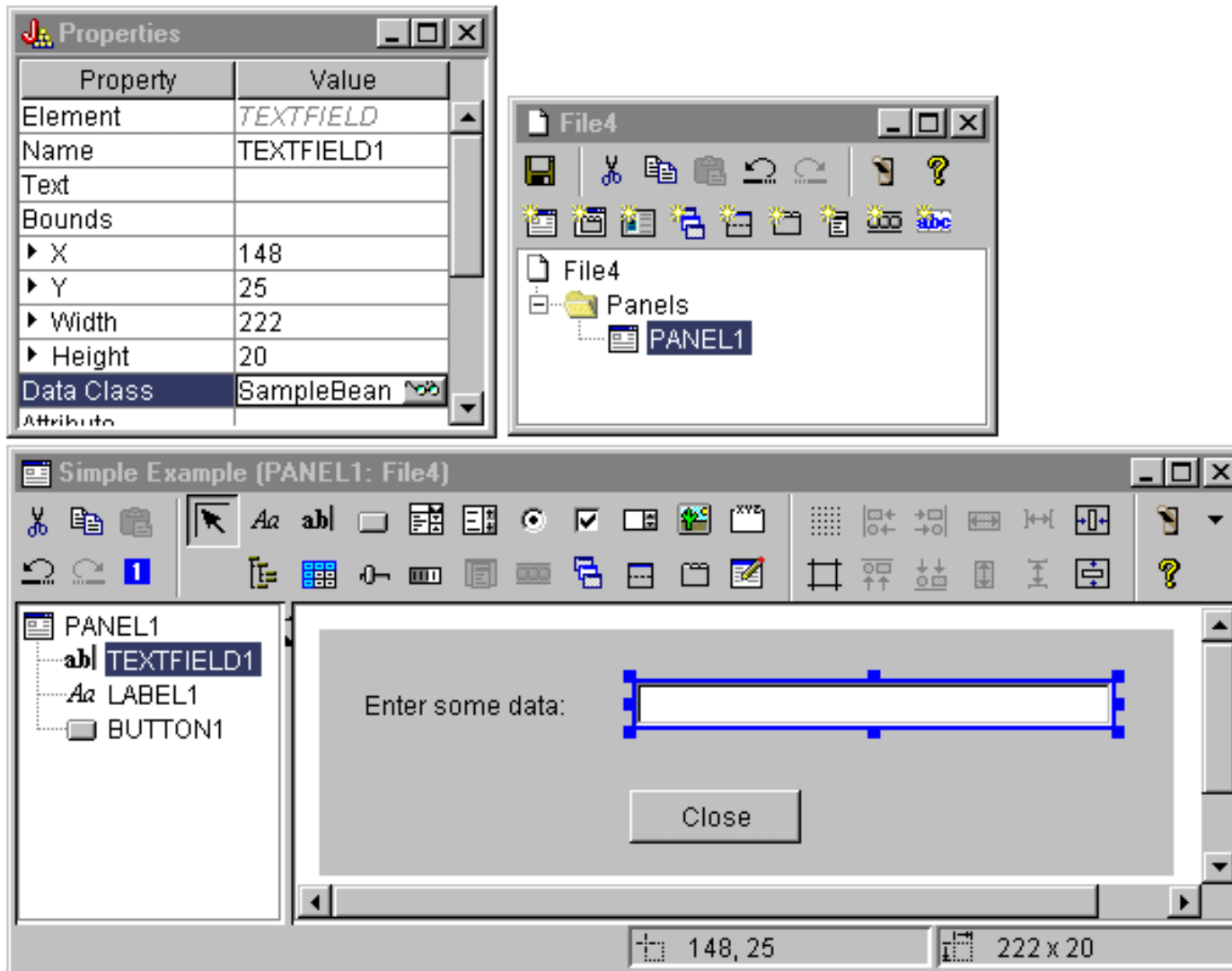
**Figura 2: finestre GUI Builder: modifica del testo nella finestra Proprietà**



### Campo di testo

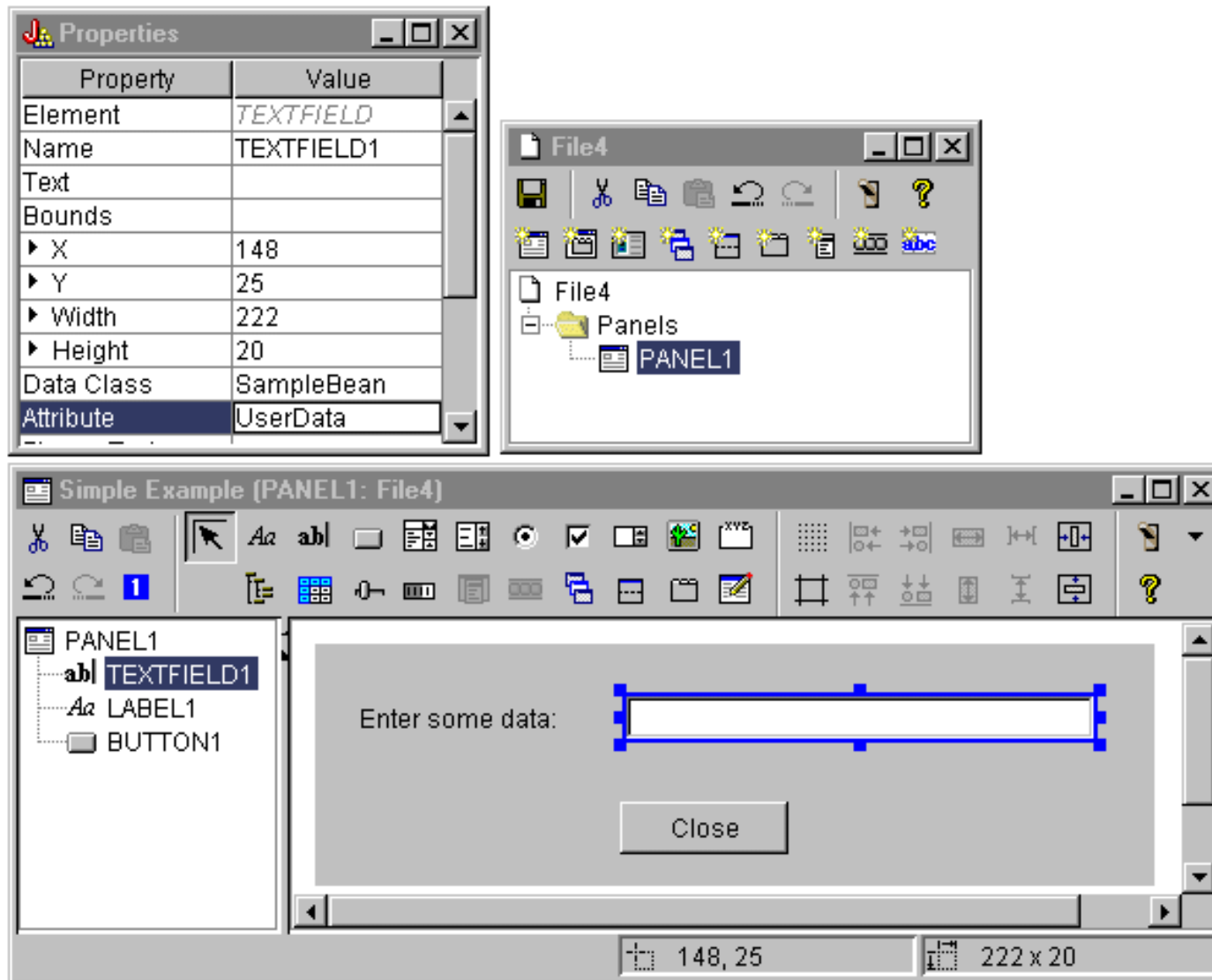
Il campo di testo contiene dati, quindi, è possibile impostare numerose proprietà che consentono al GUI Builder di eseguire del lavoro aggiuntivo. Per questo esempio, si imposta la proprietà Classe dati sul nome di una classe bean denominata **SampleBean**. Questo databean fornirà i dati per questo campo 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**.

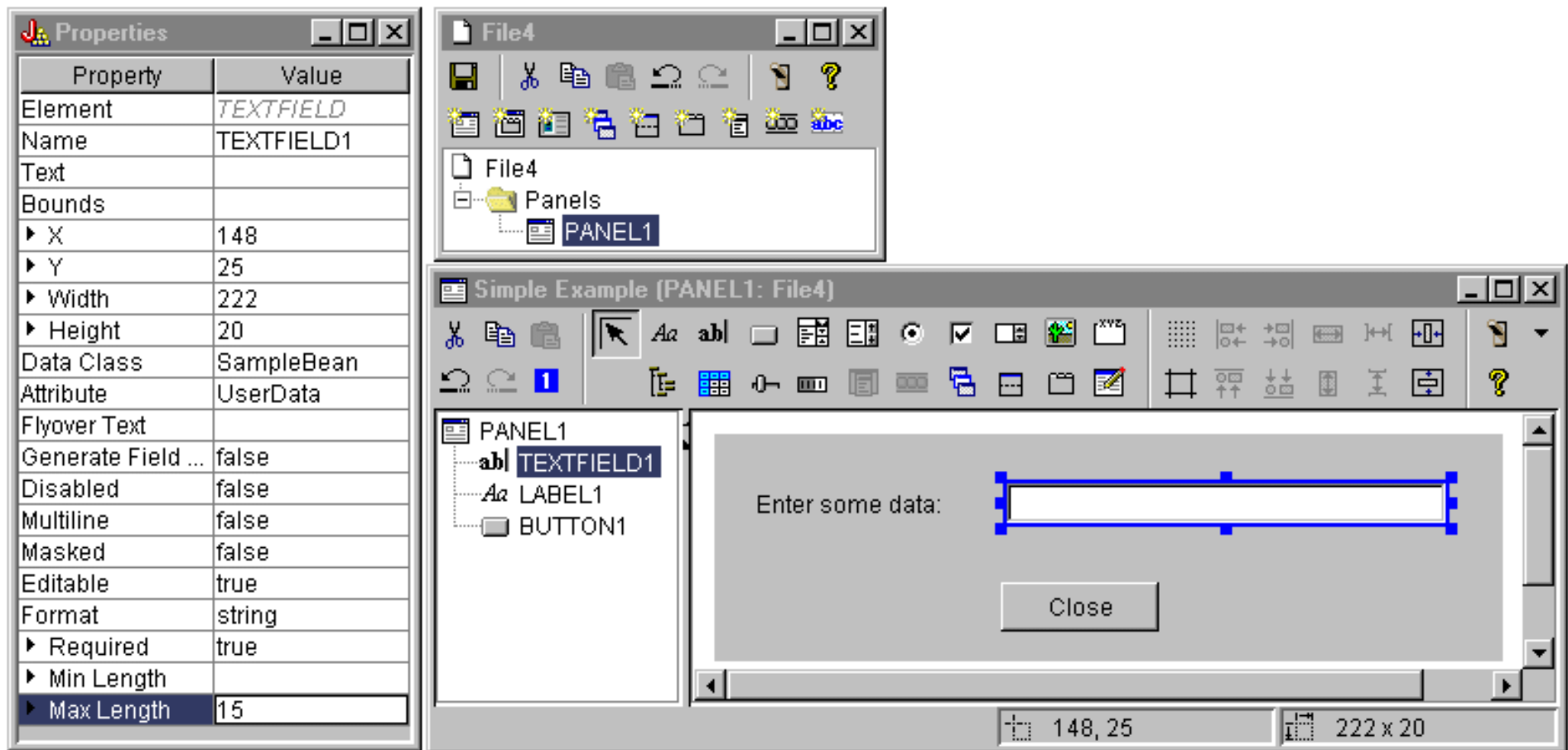
**Figura 4: finestre GUI Builder: impostazione della proprietà Attributo**



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.

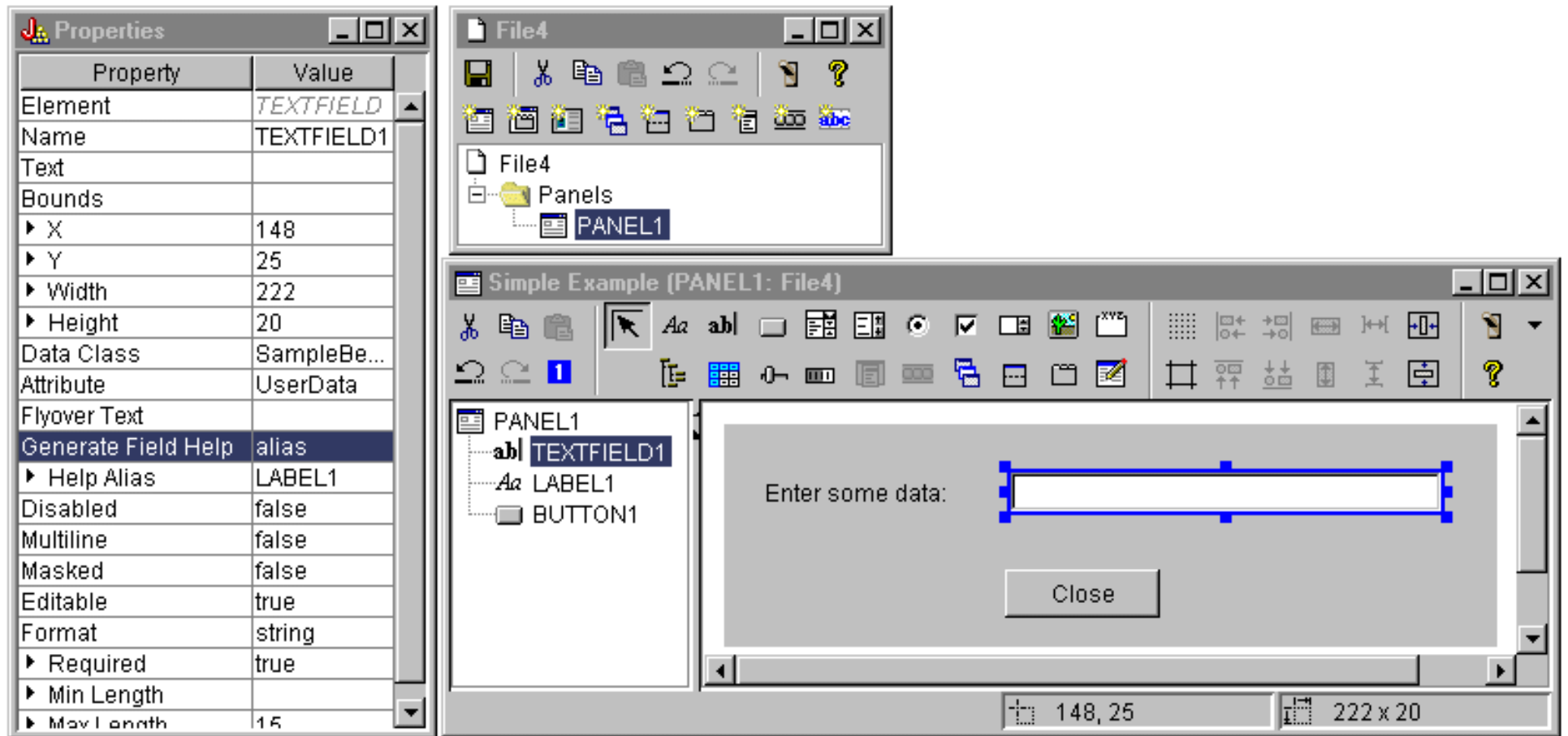
**Figura 5: finestre GUI Builder: impostazione della lunghezza massima del campo di testo**



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**

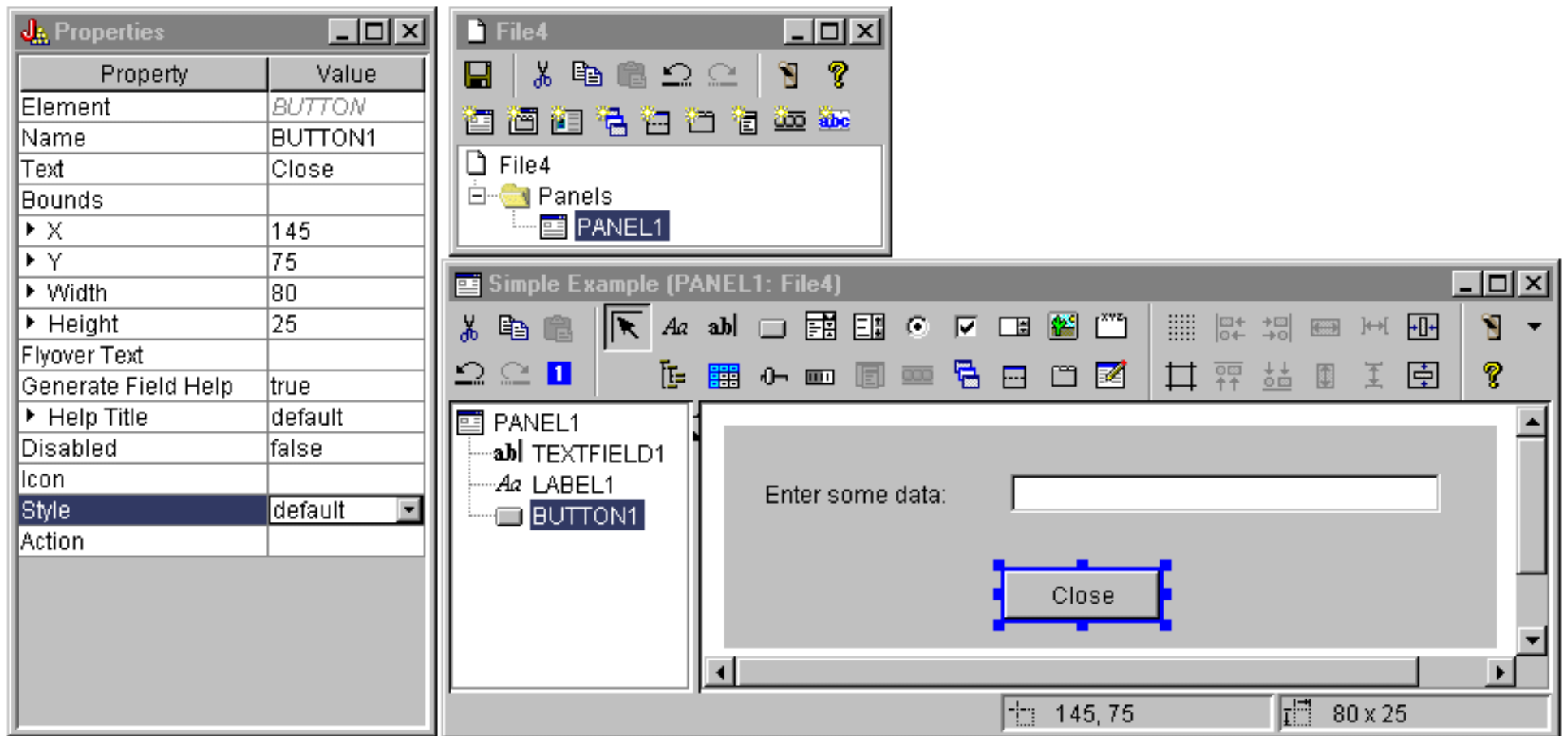




### Pulsante

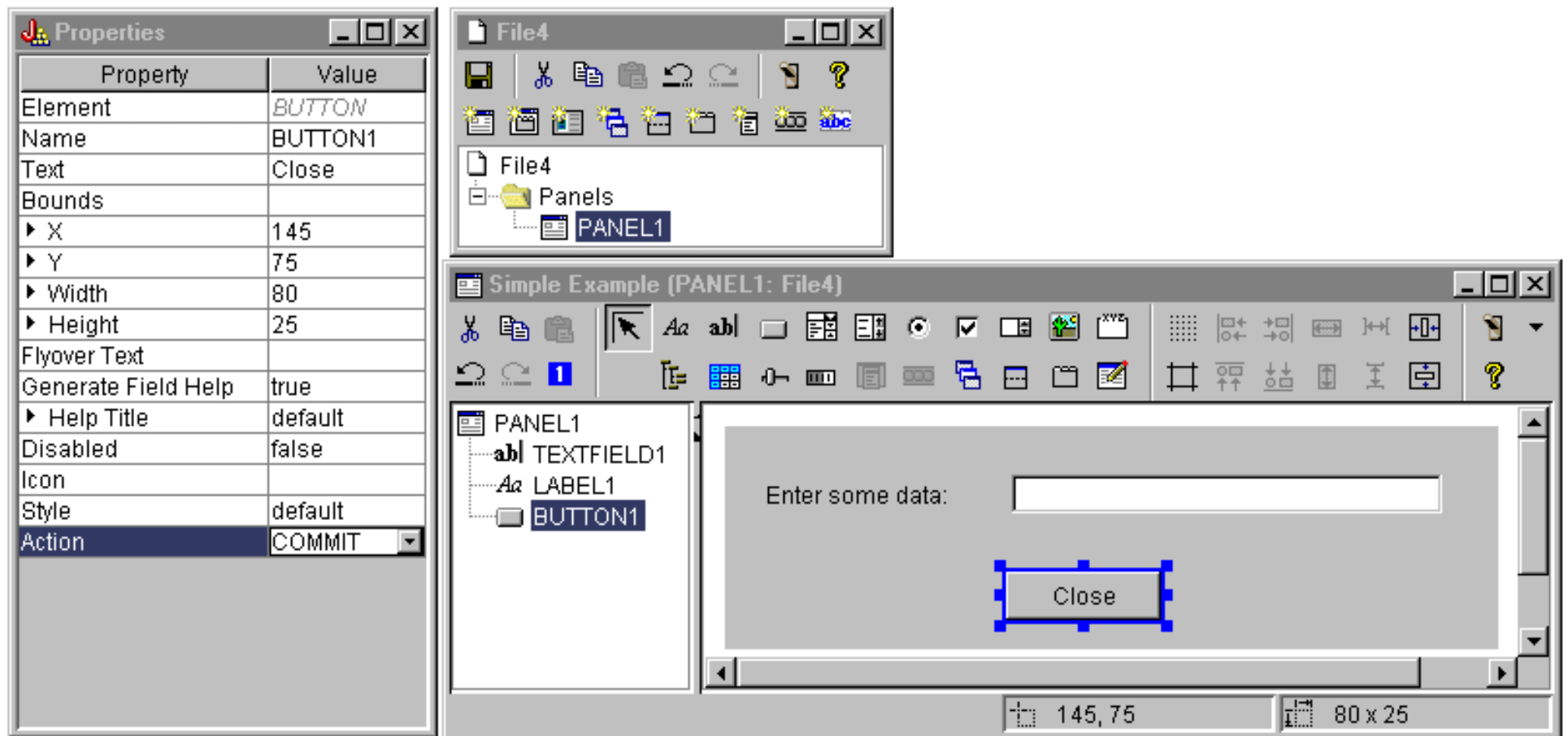
Modificare la proprietà di stile per assegnare al pulsante le caratteristiche predefinite.


**Figura 7: finestre GUI Builder: impostazione della proprietà dello stile per assegnare al pulsante le caratteristiche predefinite**



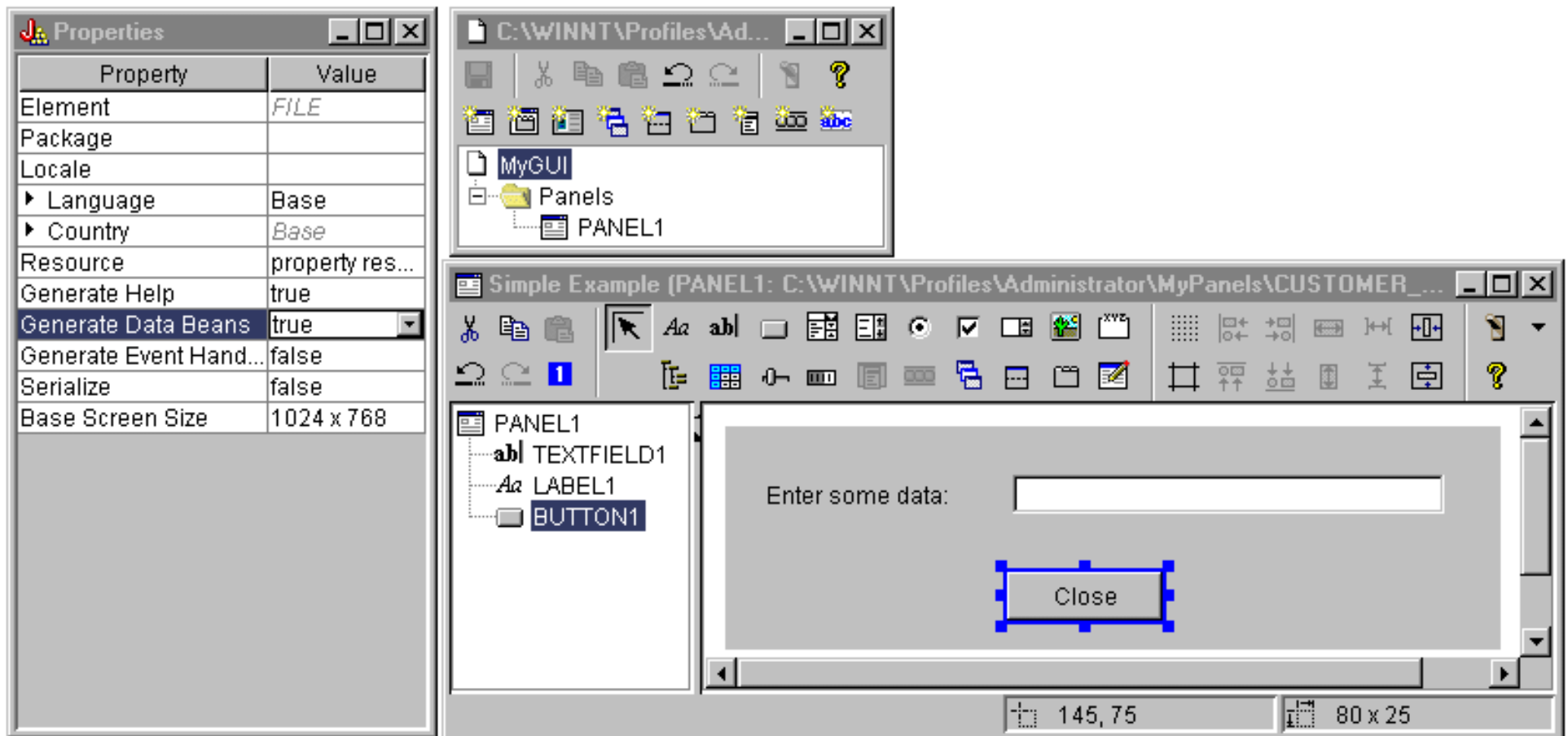
Impostare la proprietà ACTION su COMMIT, per fare in modo che il metodo `setUserData` sul bean venga chiamato quando viene selezionato il pulsante.

**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 ed il java bean**



## 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 (Panel Definition Markup Language). Dal momento che si utilizza PDML solo tramite gli strumenti forniti dal Graphical Toolbox, non è necessario comprendere nel dettaglio il formato di questo file:

```
<!-- Generated by 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 getter e setter 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:

- [Creare il documento aiuto](#)
- [Modificare i documenti di aiuto creati dal GUI Builder](#)

## Creare l'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. Lista degli oggetti di dati che forniscono 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 sono verificati degli errori, 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.

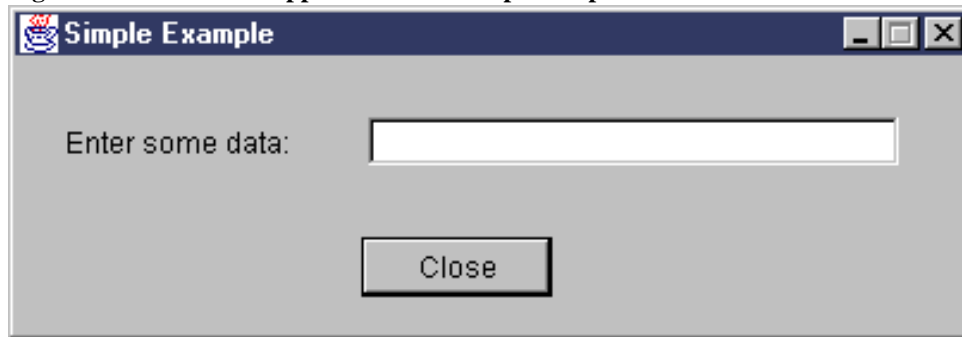
La classe `com.ibm.as400.ui.framework.java.PanelManager` fornisce l'API per visualizzare finestre e finestre di dialogo autonome. Il nome del file PDML come viene fornito nel programma di creazione, viene considerato come nome della risorsa dal Graphical Toolbox, l'indirizzo, il file ZIP o il file JAR 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.

## Eeguire un'applicazione

Questo è un esempio di come appare la finestra quando l'applicazione viene compilata ed eseguita:

**Figura 10: finestra dell'applicazione di 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 [Utilizzare 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.

## Esempio

Supponiamo 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.isAssignableFrom(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 */ }
}
```

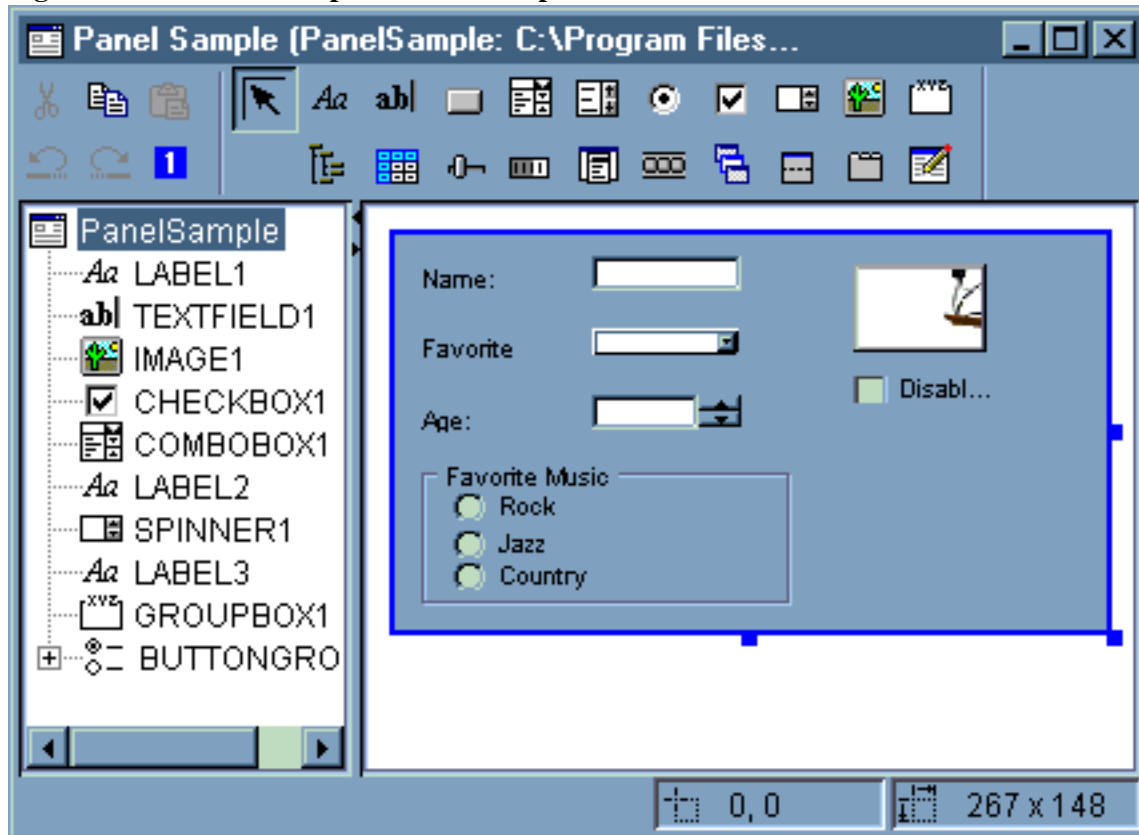
# Creare un pannello con il GUI Builder

La creazione di un pannello con il GUI Builder è semplice. Dalla barra dei menu sulla finestra GUI Builder principale, 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 il 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.

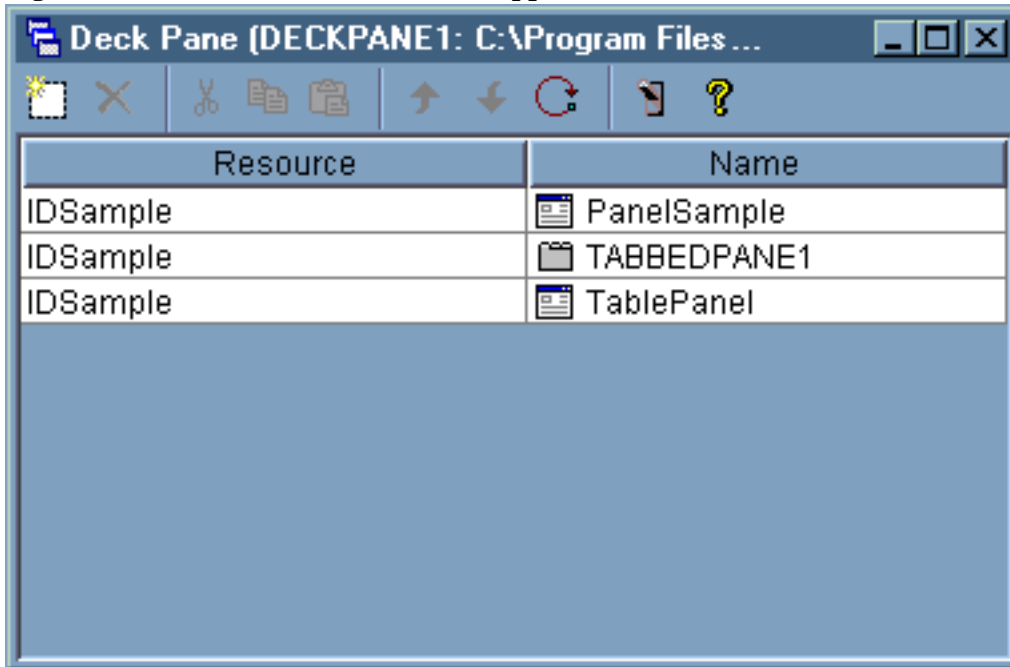
## Creare un pannello sovrapposto con il GUI Builder

Il GUI Builder crea un semplice pannello sovrapposto. Dalla barra del menu sulla finestra GUI Builder, selezionare **File --> Nuovo file**.

Dalla barra dei menu nella finestra **File** del GUI Builder, fare clic sul pulsante **Inserisci pannello sovrapposto** per visualizzare un programma di creazione pannello dove è possibile inserire le componenti per il pannello sovrapposto. Nell'esempio seguente, vengono aggiunti tre componenti.

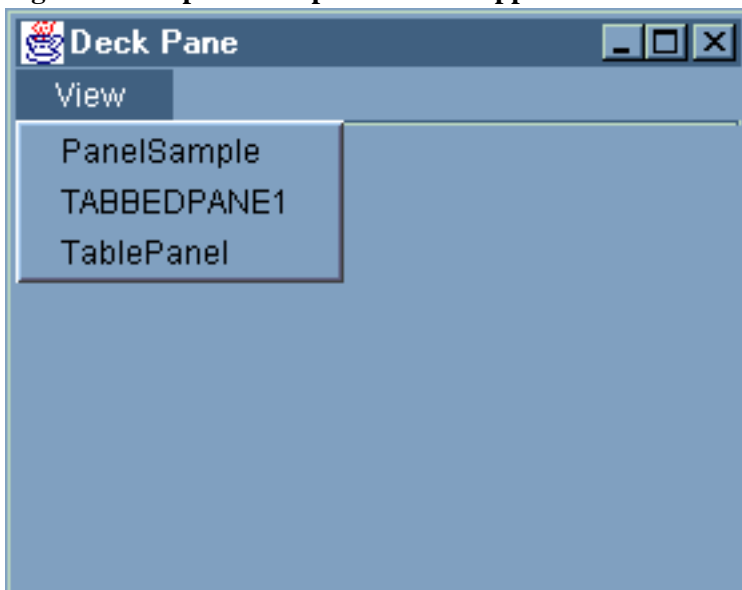


**Figura 1: creazione di un Pannello sovrapposto con il GUI Builder**



Dopo aver creato il pannello sovrapposto, fare clic sul pulsante **Anteprima** per visualizzarlo in anteprima. Si visualizza un pannello sovrapposto fino a che non si seleziona il menu **Visualizza**.

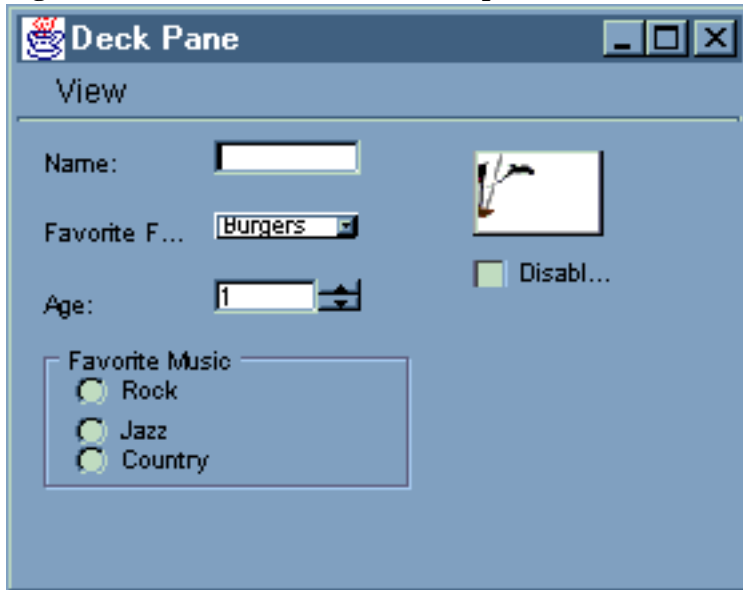
**Figura 2: anteprima del pannello sovrapposto con il GUI Builder**



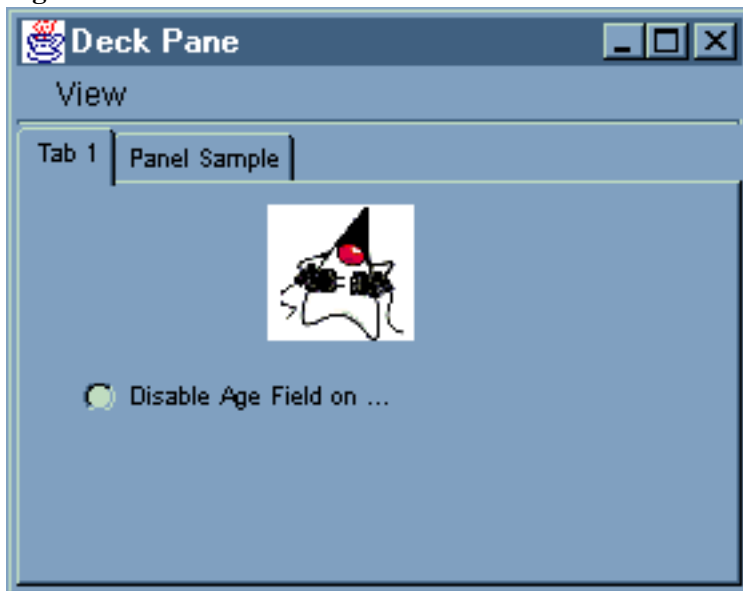
Dal menu **Visualizza** del pannello sovrapposto, selezionare il componente che si desidera visualizzare. Per questo esempio, si può scegliere di visualizzare il PanelSample, TABBEDPANE1 o il TablePanel. Le seguenti figure

illustrano come vengono visualizzati questi componenti.

**Figura 3: visualizzazione di PanelSample con il GUI Builder**

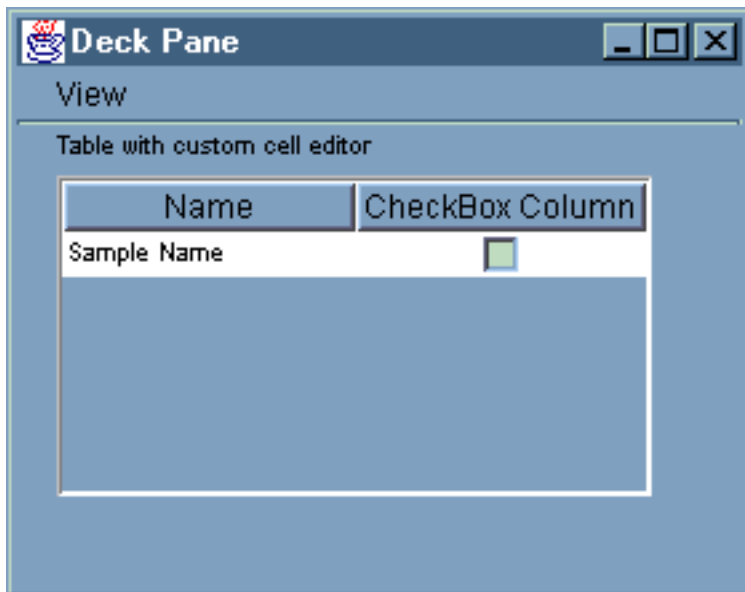


**Figura 4: visualizzazione di TABBEDPANE1 con il GUI Builder**




**Figura 5: visualizzazione di TablePanel con il GUI Builder**



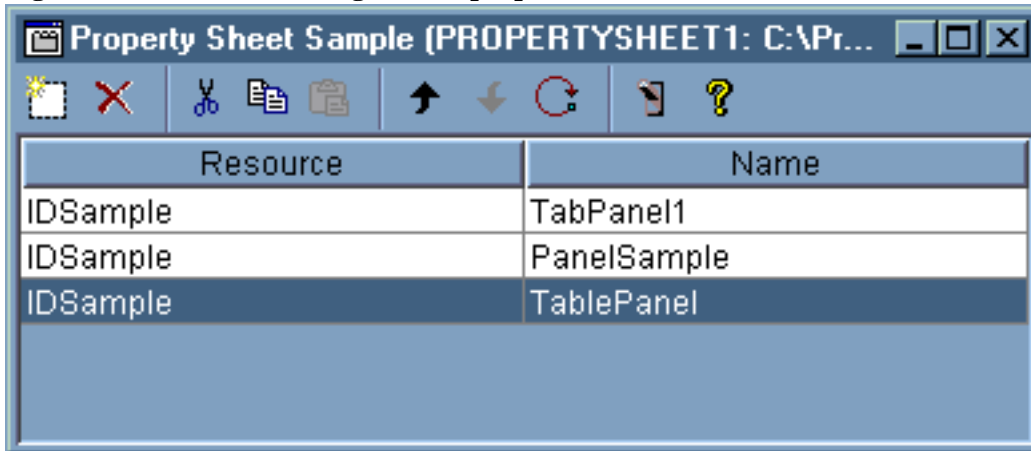


## Creare un foglio delle proprietà con il GUI Builder

Il GUI Builder semplifica la creazione di un foglio delle proprietà. Dalla barra dei menu sulla finestra principale del GUI Builder, selezionare **File --> Nuovo file**.

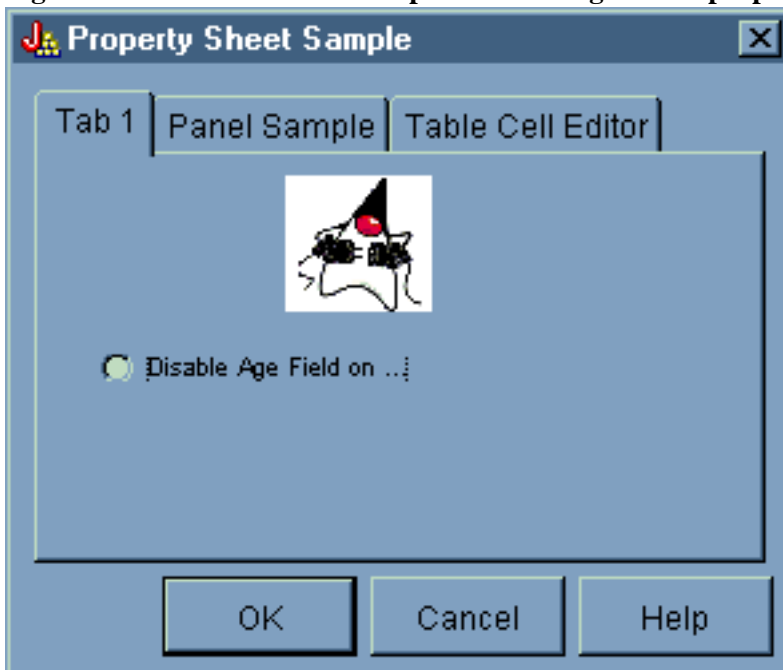
Dalla barra dei menu nella finestra **File** del GUI Builder, fare clic sull'icona Inserisci foglio proprietà  per visualizzare un programma di creazione del pannello in cui si è possibile inserire i componenti del foglio delle proprietà.

**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**



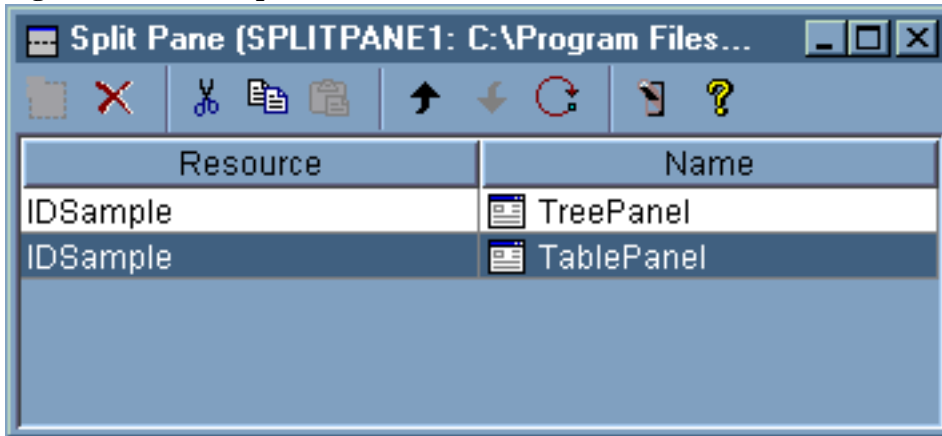
## Creare un pannello suddiviso con il GUI Builder


Il GUI Builder fa sì che la creazione di un pannello suddiviso sia semplice. Dalla barra dei menu sulla finestra principale della GUI Builder, selezionare **File --> Nuovo file**.

Dalla barra dei menu della finestra **File** del GUI Builder, fare clic sul pulsante strumento **Inserisci pannello suddiviso**

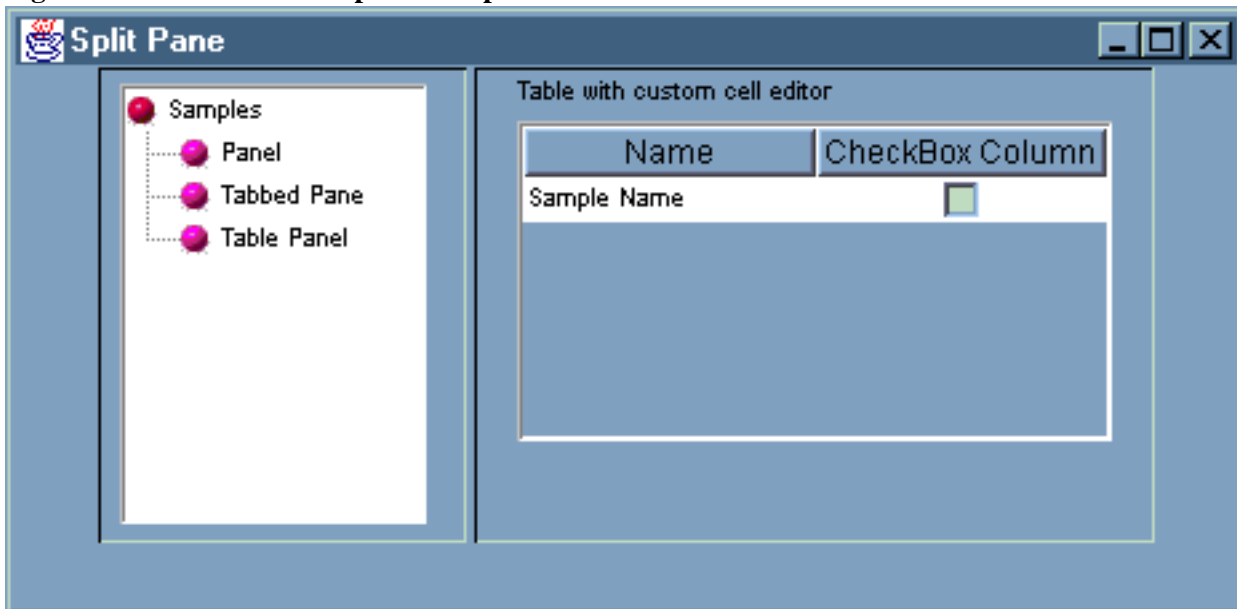
 per visualizzare un programma di creazione del pannello in cui è possibile inserire i componenti desiderati. Nell'esempio che segue, vengono aggiunti due componenti.

**Figura 1: creare un pannello suddiviso con il GUI Builder**




Dopo aver creato il pannello suddiviso, fare clic sul pulsante strumento **Anteprima**  per visualizzare l'anteprima, così come mostrato dalla Figura 2.

**Figura 2: visualizzare l'anteprima del pannello suddiviso con il GUI Builder**

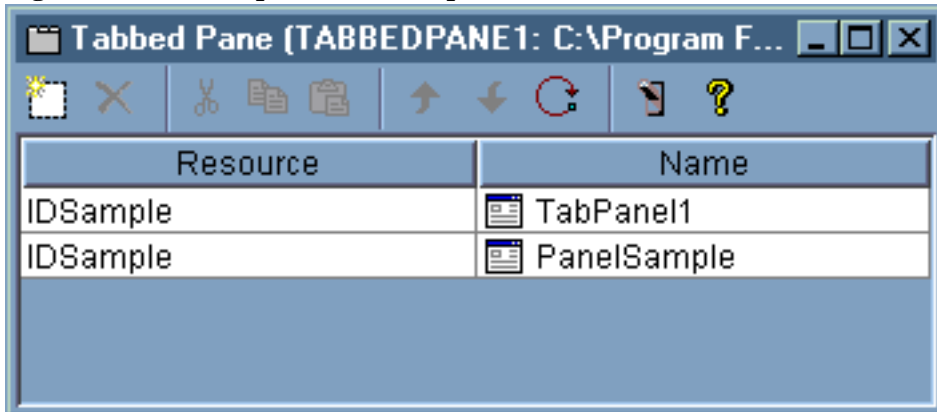



## Creare un pannello con separatori con il GUI Builder

Il GUI Builder fa sì che la creazione di un pannello con separatori sia semplice. Dalla barra dei menu sulla finestra principale del GUI Builder, selezionare **File --> Nuovo file**.

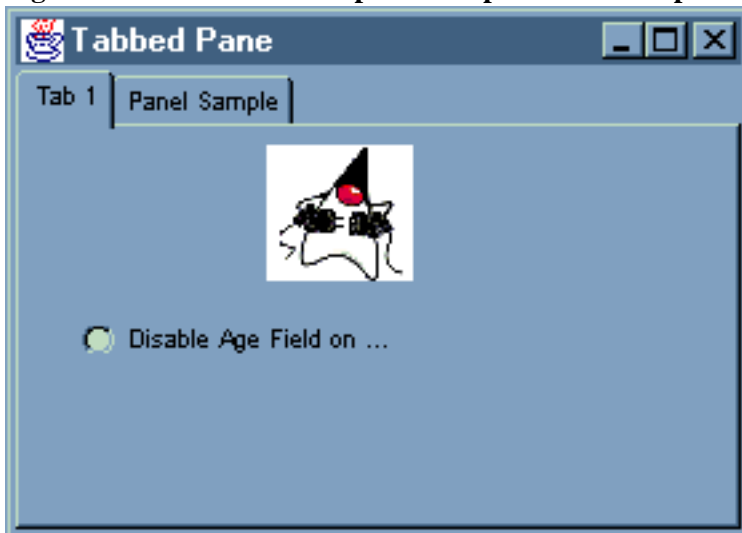
Dalla barra dei menu nella finestra **File** del Gui Builder, fare clic sull'icona **Inserisci pannello con separatori**  per visualizzare un programma di creazione del pannello in cui è possibile inserire i componenti per il pannello con separatori. Nell'esempio che segue, vengono aggiunti due componenti.

**Figura 1: creare un pannello con separatori con il GUI Builder**



Dopo aver creato il pannello con separatori, fare clic sul pulsante strumento **Anteprima**  per visualizzarlo.

**Figura 2: visualizzare l'anteprima del pannello con separatori con il GUI Builder**



## Creare un wizard con il GUI Builder

Il GUI Builder semplifica la creazione di un'interfaccia wizard. Dalla barra del menu sulla finestra GUI Builder, selezionare **File --> Nuovo file**.


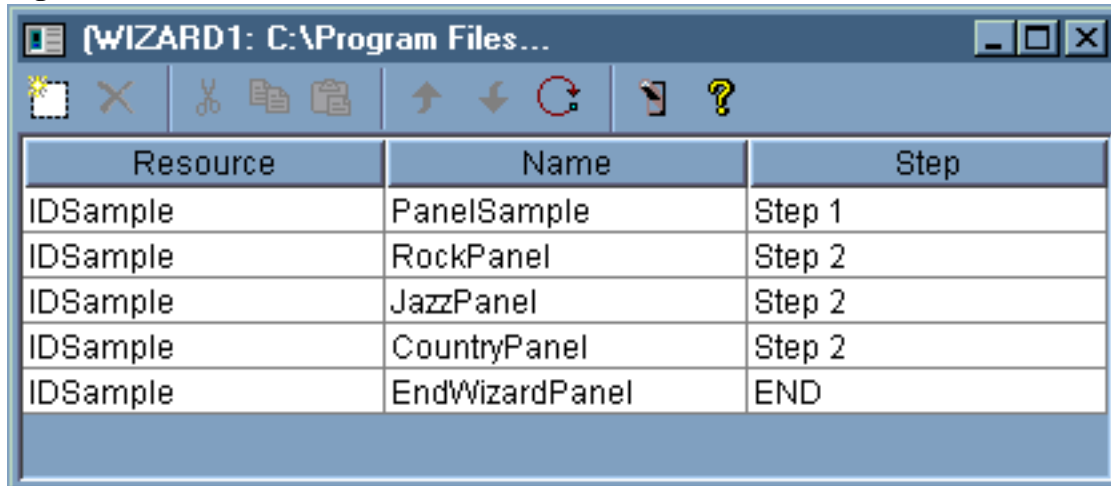
Dalla barra dei menu sulla finestra **File** del GUI Builder, fare clic sul pulsante della barra degli Inserisci wizard  per visualizzare un programma di creazione del pannello da cui è possibile aggiungere pannelli al wizard.

Figura 1: creazione di un wizard con il GUI Builder



Resource	Name	Step
IDSample	PanelSample	Step 1
IDSample	RockPanel	Step 2
IDSample	JazzPanel	Step 2
IDSample	CountryPanel	Step 2
IDSample	EndWizardPanel	END


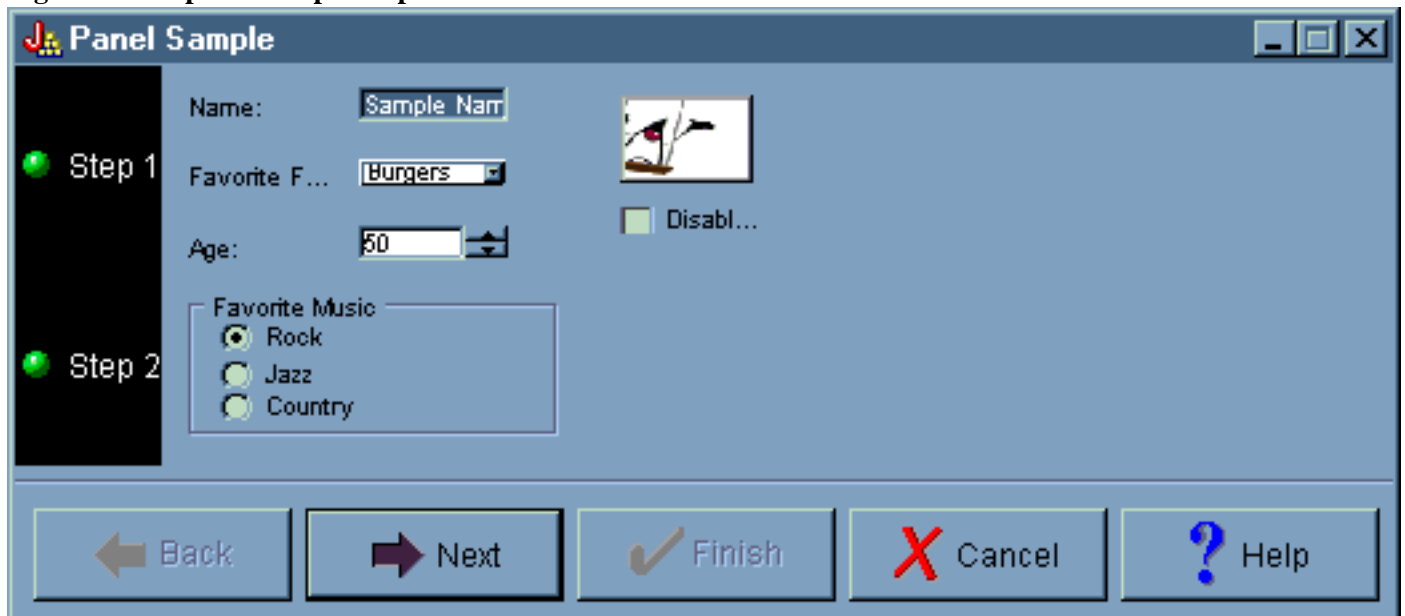
Dopo aver creato un wizard, utilizzare il pulsante dello strumento **Anteprima**  per visualizzarlo. La figura 2 mostra il primo pannello visualizzato per questo esempio.

Figura 2: anteprima del primo pannello wizard con il GUI Builder



**Panel Sample**

Step 1

Name:

Favorite F...:

Age:

Disabl...

Favorite Music

Rock

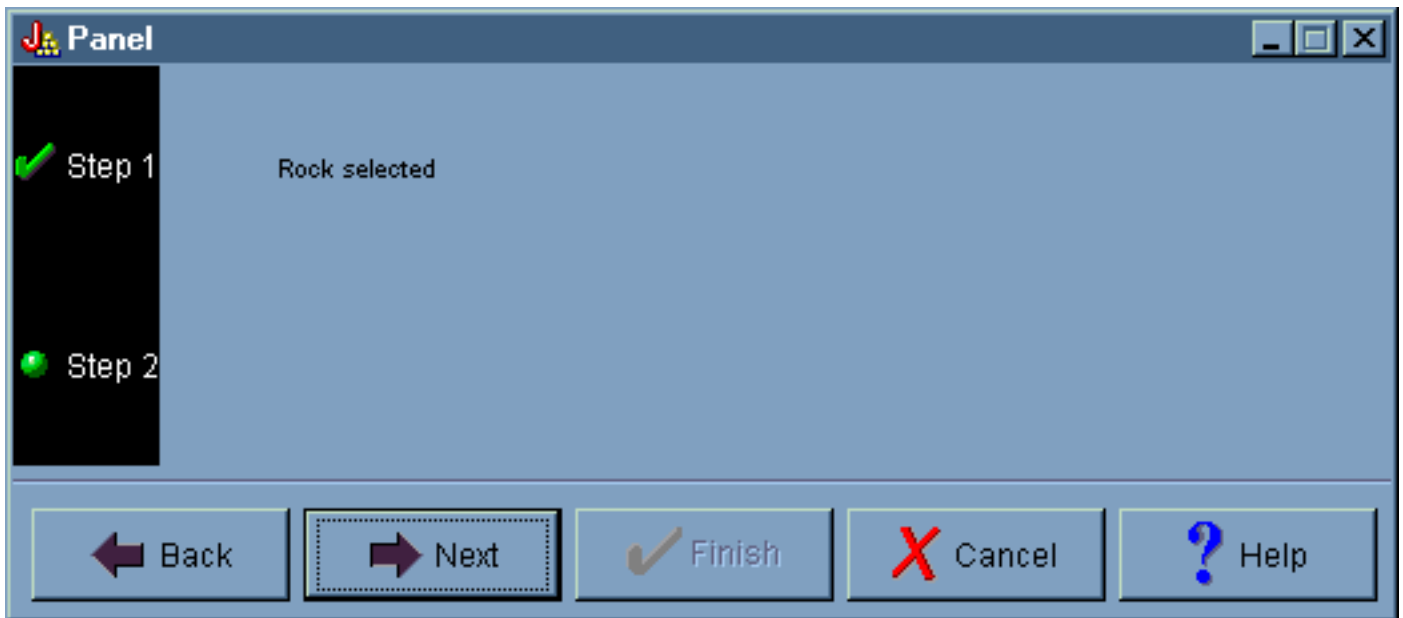
Jazz

Country

Back Next Finish Cancel Help

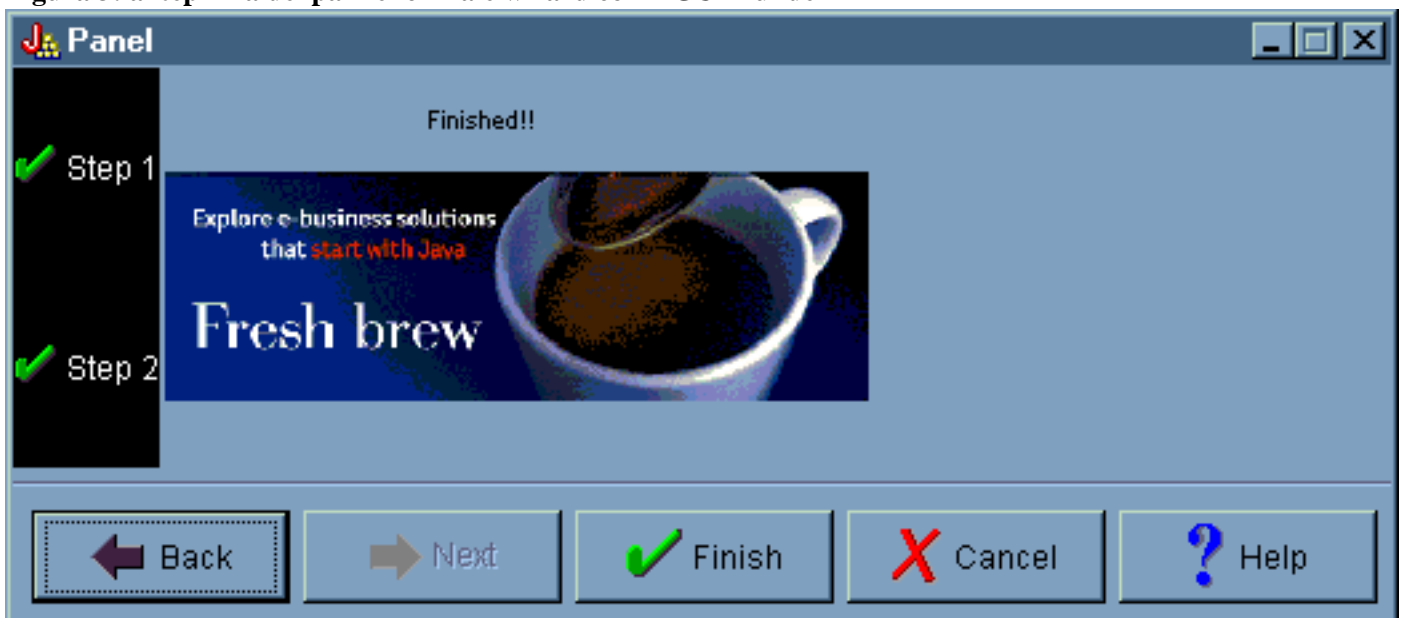
La figura 2 mostra il secondo pannello visualizzato quando l'utente seleziona **Rock** e fa clic su **Avanti**.

Figura 2: anteprima del secondo pannello wizard con il GUI Builder



Facendo clic su **Avanti** sul secondo pannello wizard viene visualizzato il pannello finale wizard, come illustrato nella Figura 3.

**Figura 3: anteprima del pannello finale wizard con il GUI Builder**

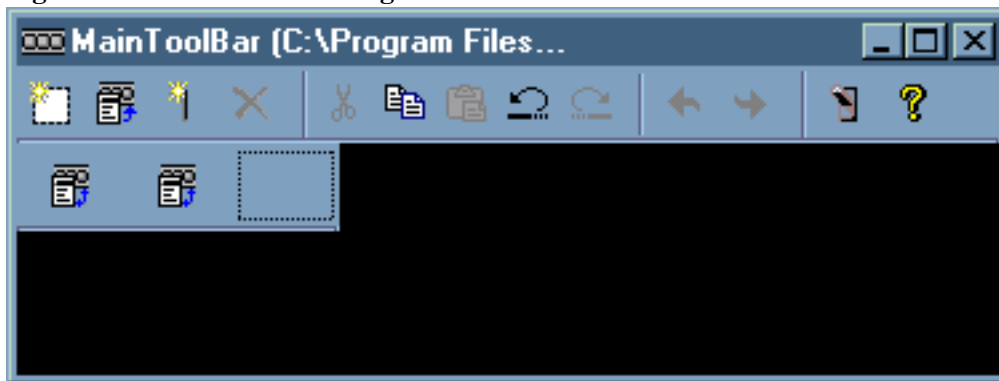



## Creare una barra degli strumenti con il GUI Builder

Il GUI Builder rende semplice la creazione di una barra degli strumenti. Dalla barra del menu sulla finestra GUI Builder, selezionare **File --> Nuovo file**.

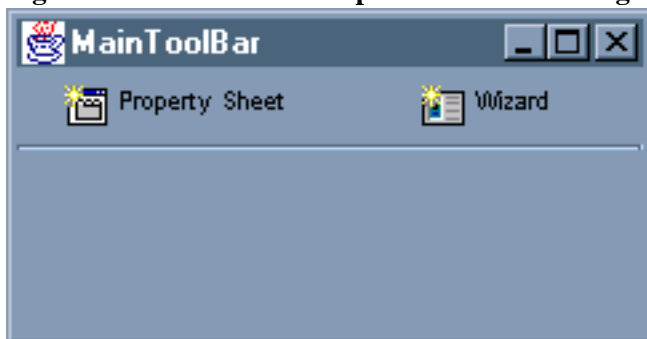
Dalla barra dei menu sulla finestra **File** del GUI Builder, fare clic sul pulsante strumento **Inserisce barra degli strumenti** per visualizzare il programma di creazione del pannello in cui è possibile inserire i componenti per la barra degli strumenti.

**Figura 1: creare una barra degli strumenti con il GUI Builder**



Dopo aver creato la barra degli strumenti, fare clic sul pulsante strumento **Anteprima**  per visualizzare l'anteprima. Per questo esempio, è possibile visualizzare il foglio delle proprietà o il wizard.

**Figura 2: visualizzare l'anteprima della barra degli strumenti con il GUI Builder**

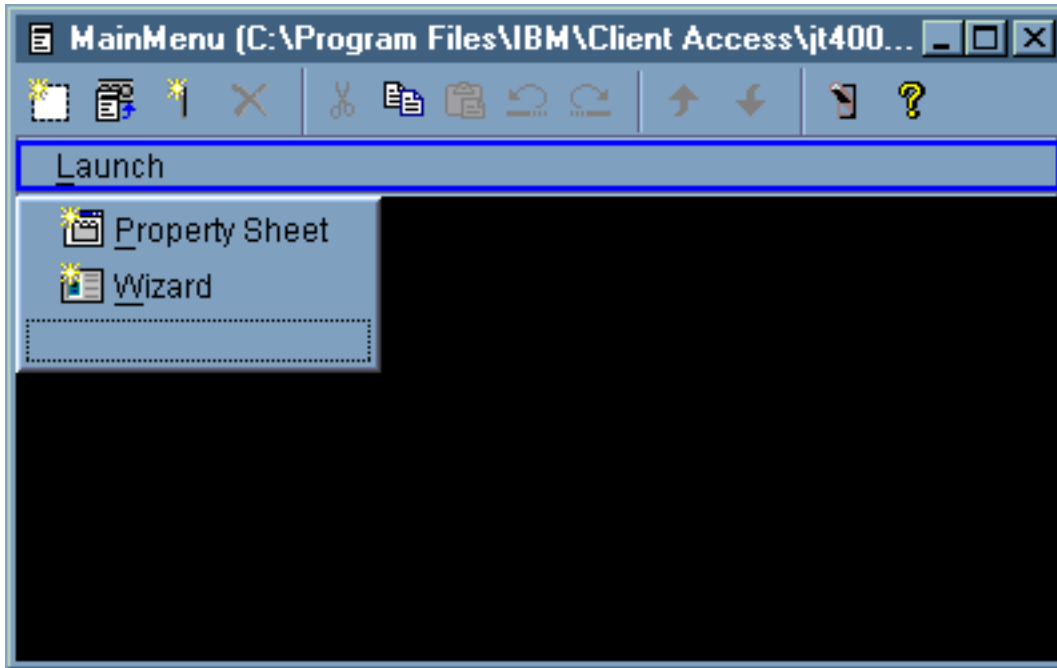



## Creare una barra dei menu con il GUI Builder

Il GUI Builder semplifica la creazione di una barra dei menu. Dalla barra del menu sulla 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**



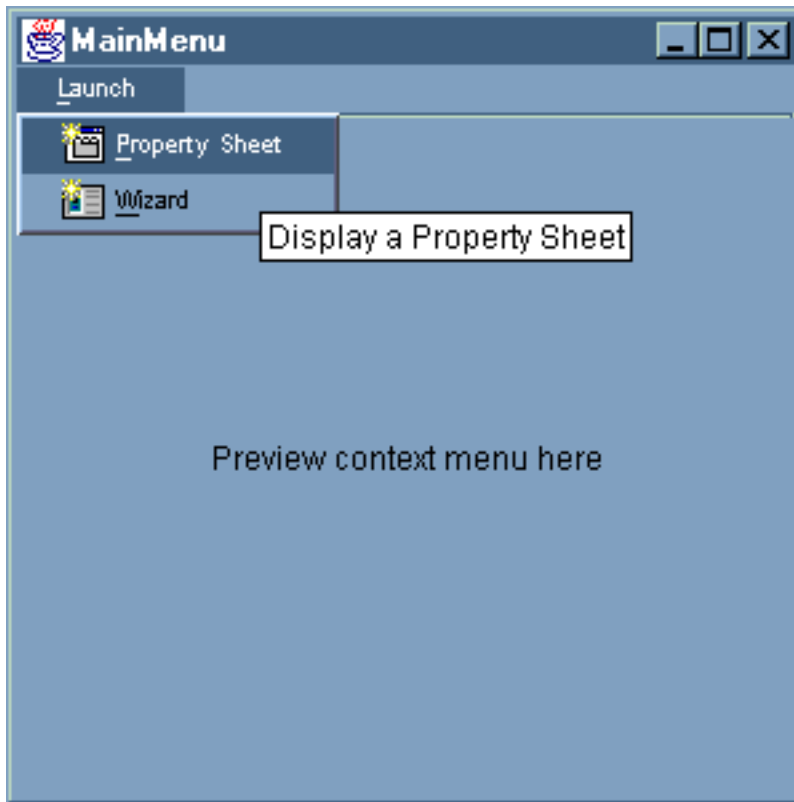
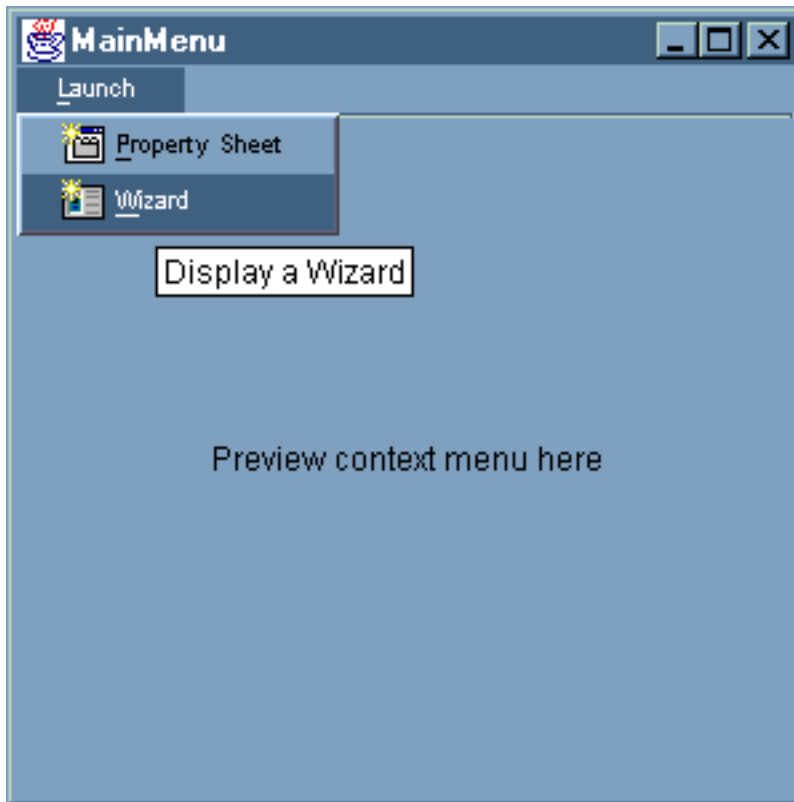


Figura 3: GUI Builder: visualizzazione del Wizard sul menu Avvio



## Esempio: creare il 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.

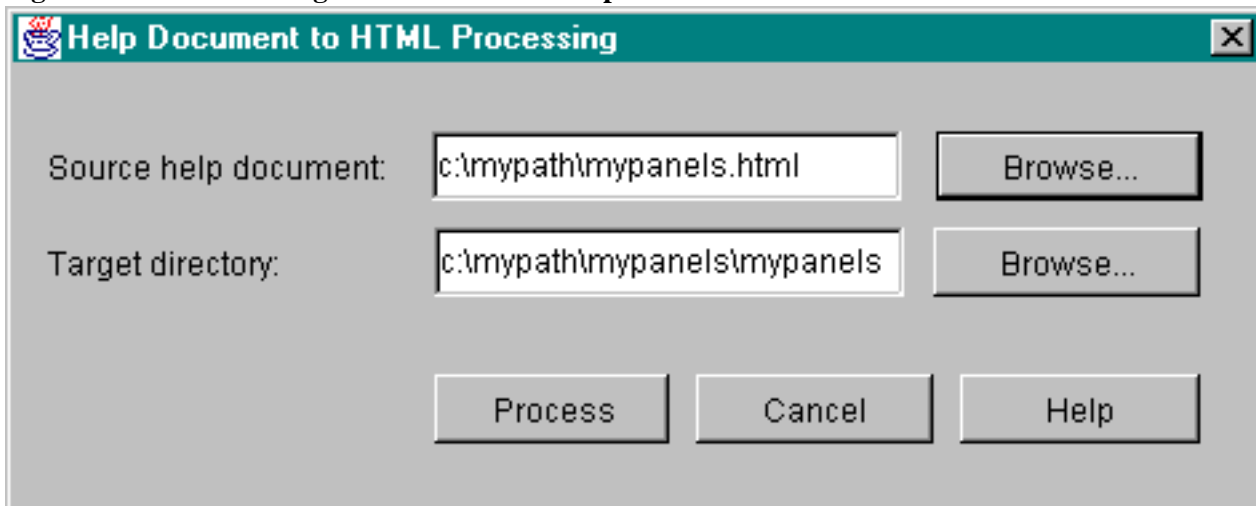
Figura 1: impostazione della proprietà Crea aiuto sul pannello delle proprietà del GUI Builder



GUI Builder crea una framework HTML denominata Documento di aiuto, che è possibile [modificare](#).

Per fare in modo che vengano utilizzati al tempo di esecuzione, gli argomenti all'interno del file PDML devono essere separati in file HTML singoli. 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: casella 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.hdoc2htmlViewer
```

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.

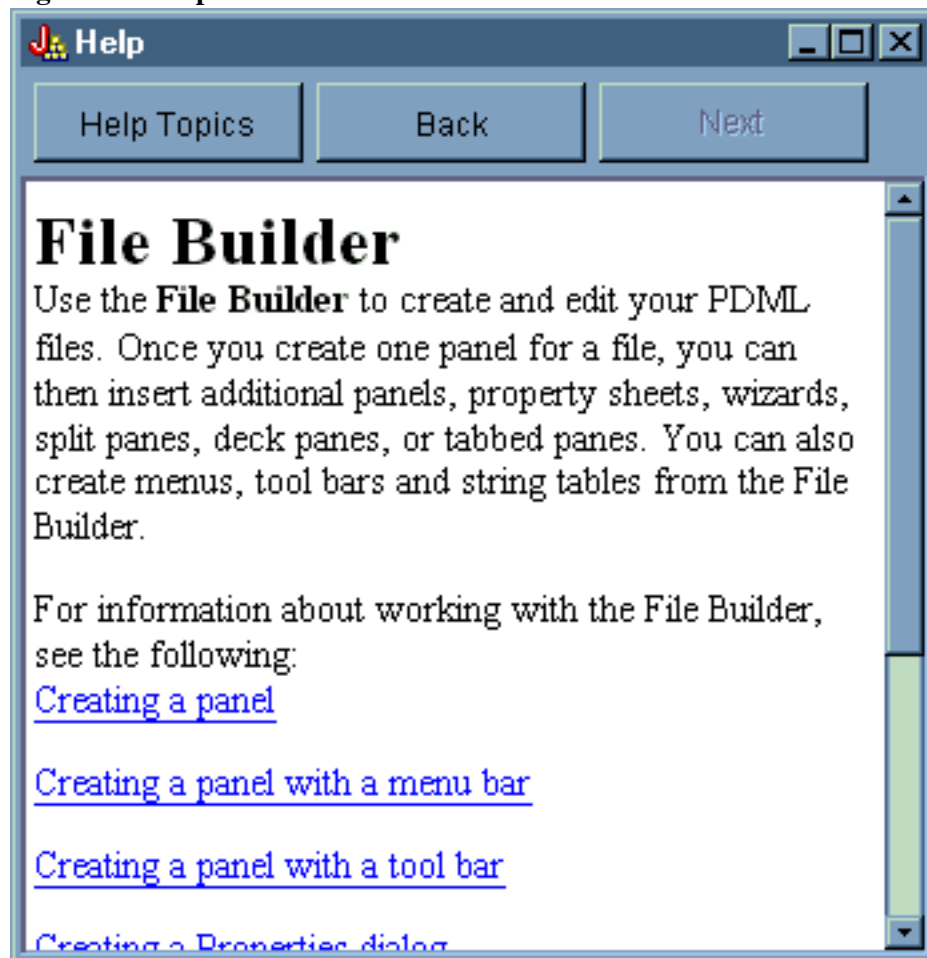
E' possibile suddividere il documento di aiuto dalla riga comandi con il comando che segue:

```
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**

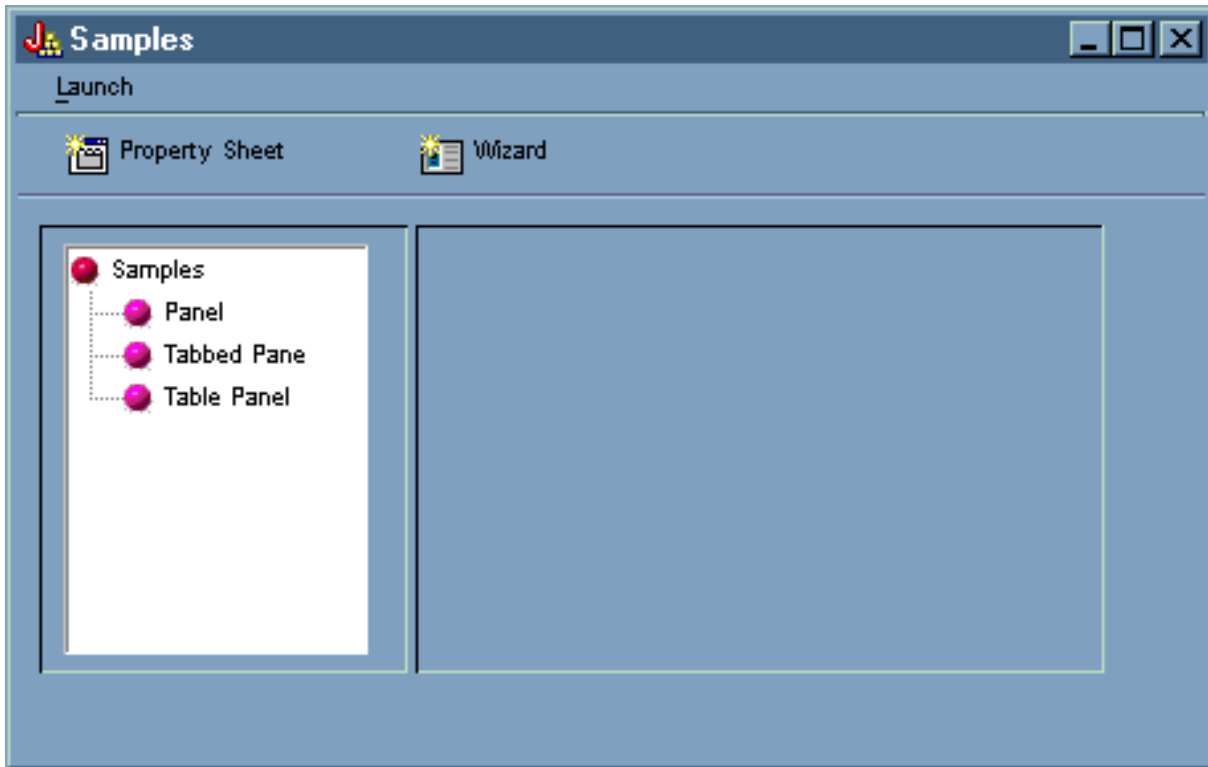


# Esempio: utilizzare il GUI Builder

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)**

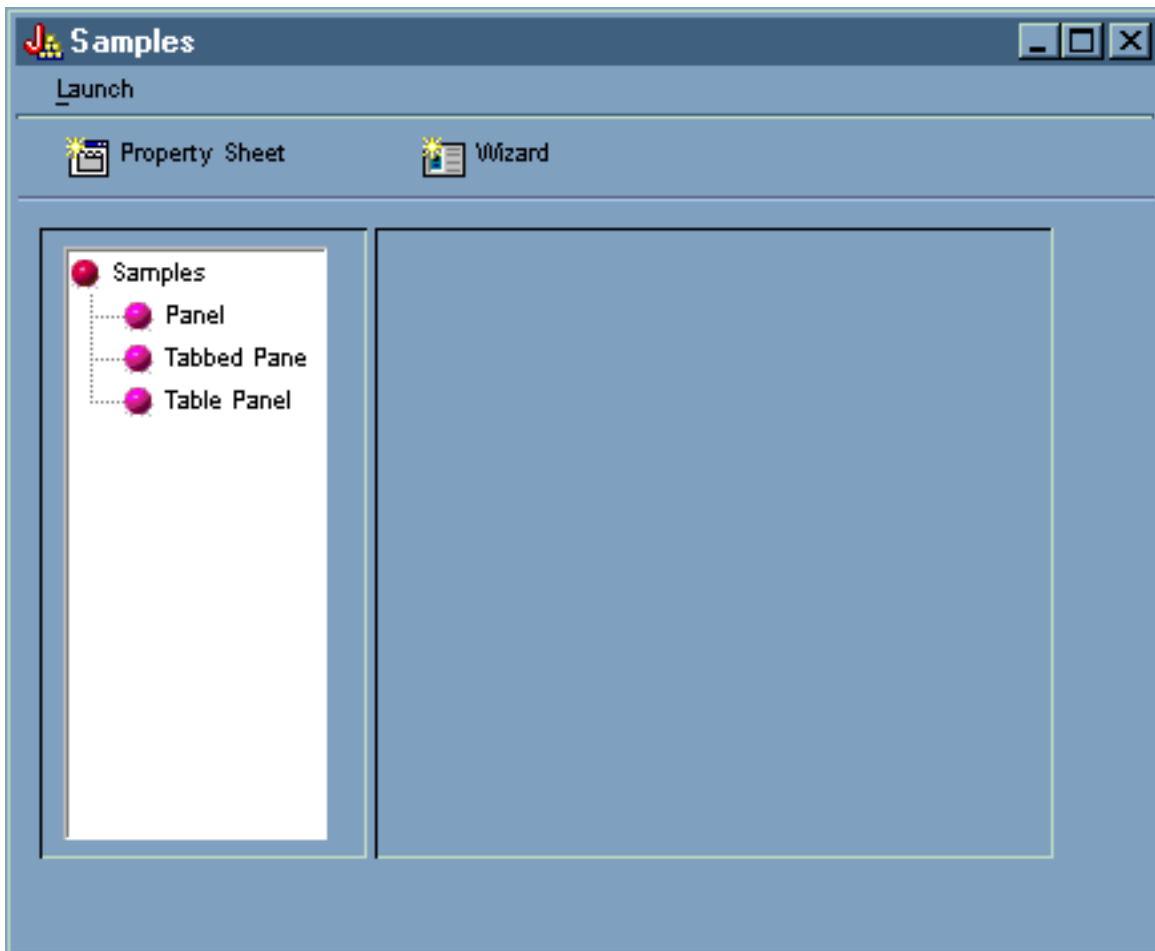
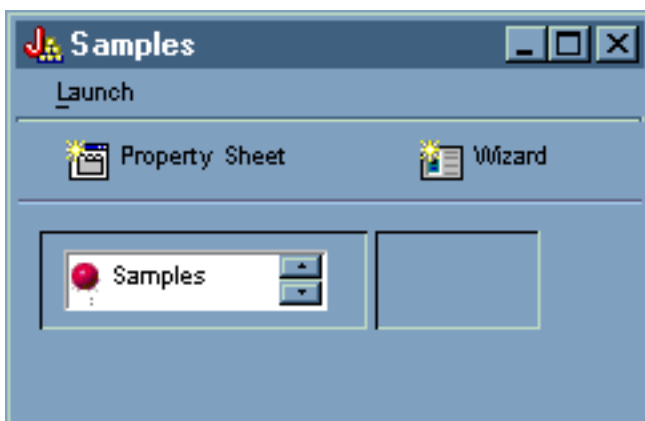


Figura 3: ridimensionamento della finestra principale dell'esempio del GUI Builder (più piccola)



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.

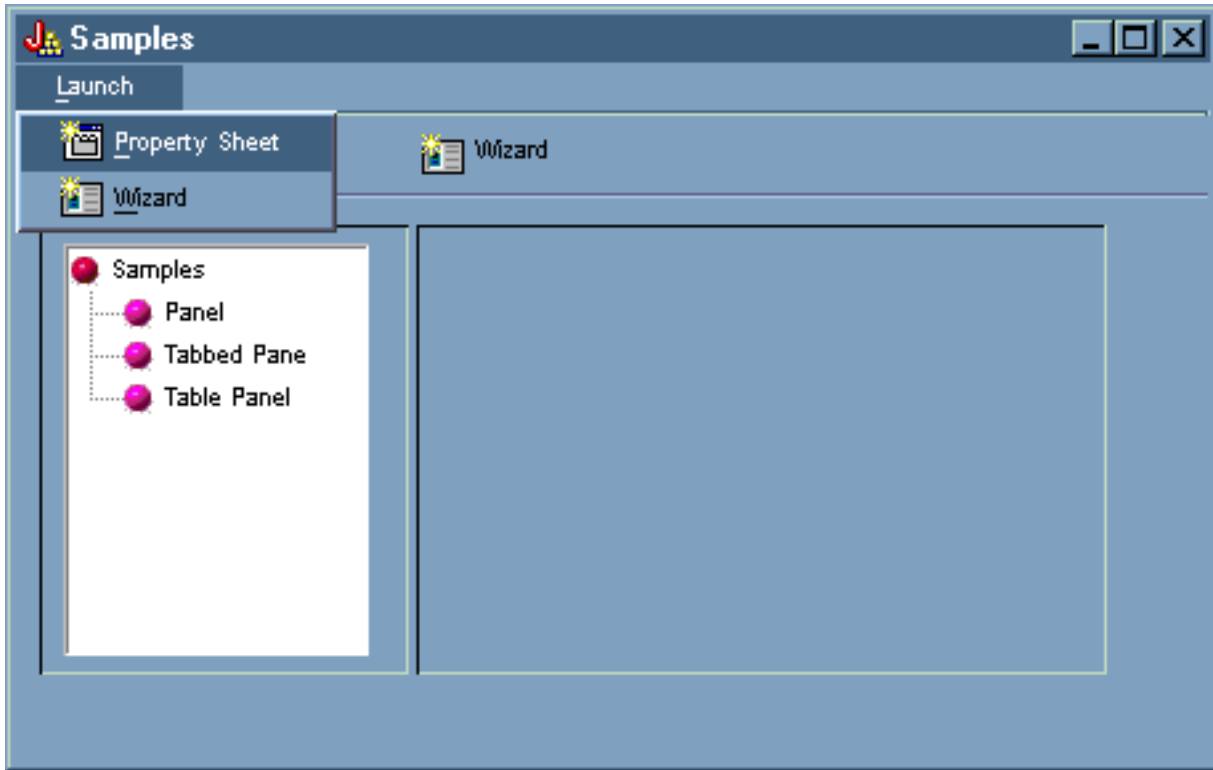
Il pannello consente di svolgere le seguenti azioni:

- [Avviare un foglio delle proprietà](#)
- [Avviare un wizard](#)
- [Visualizzare i campi elencati nel pannello di sinistra](#)

## Avviare un foglio delle proprietà

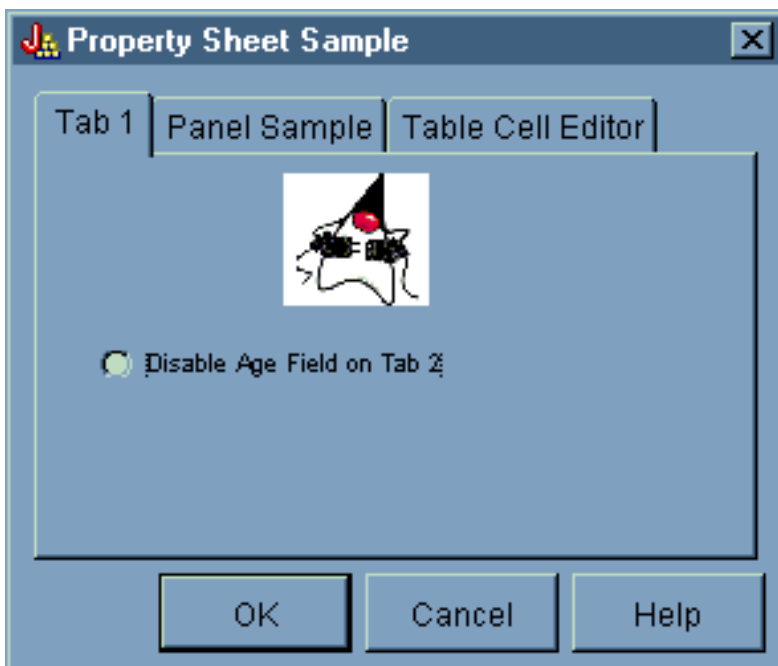
E' possibile avviare il foglio delle proprietà facendo clic sul pulsante della barra degli strumenti Foglio delle proprietà o utilizzando il menu **Avvia**. La possibilità di scegliere tra la barra degli strumenti e il menu consente di visualizzare collegamenti alle voci di menu. La figura 4 mostra il **Foglio delle proprietà** selezionato dal menu **Avvia** sulla finestra principale dell'esempio del GUI Builder.

Figura 4: selezionare Foglio delle proprietà dal menu Avvia



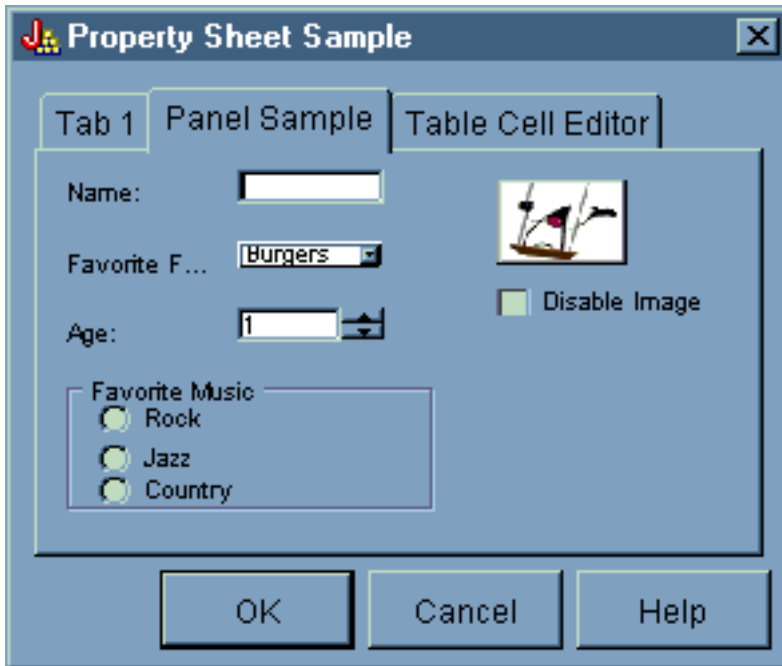
La selezione del **Foglio delle proprietà** visualizza il pannello nella figura 5.

Figura 5: finestra di dialogo Campione foglio delle proprietà

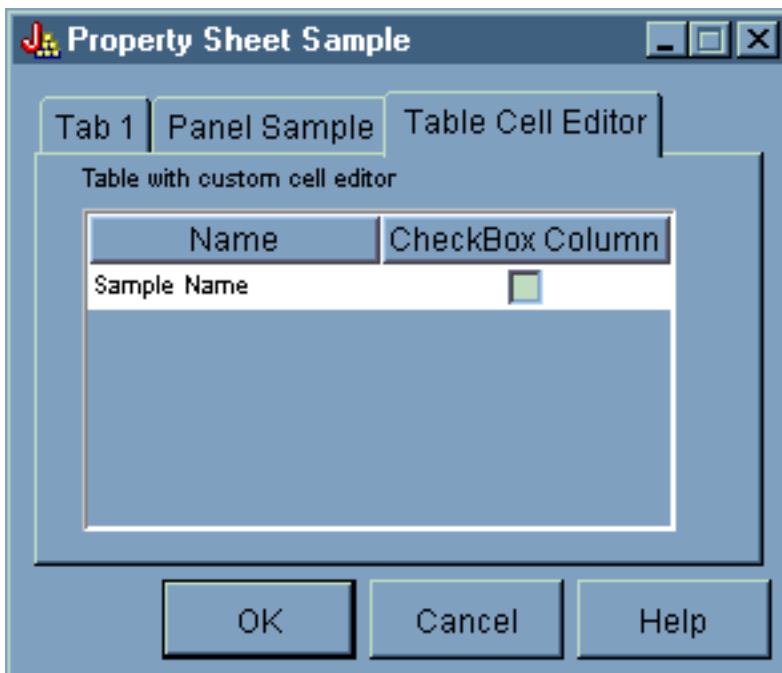


Quando appare per la prima volta il Campione foglio delle proprietà, il separatore 1 viene visualizzato per impostazione predefinita. Le figure 6 e 7 mostrano come la visualizzazione del pannello si modifichi quando si selezionano altri separatori.

**Figura 6: selezione del separatore Campione del pannello**



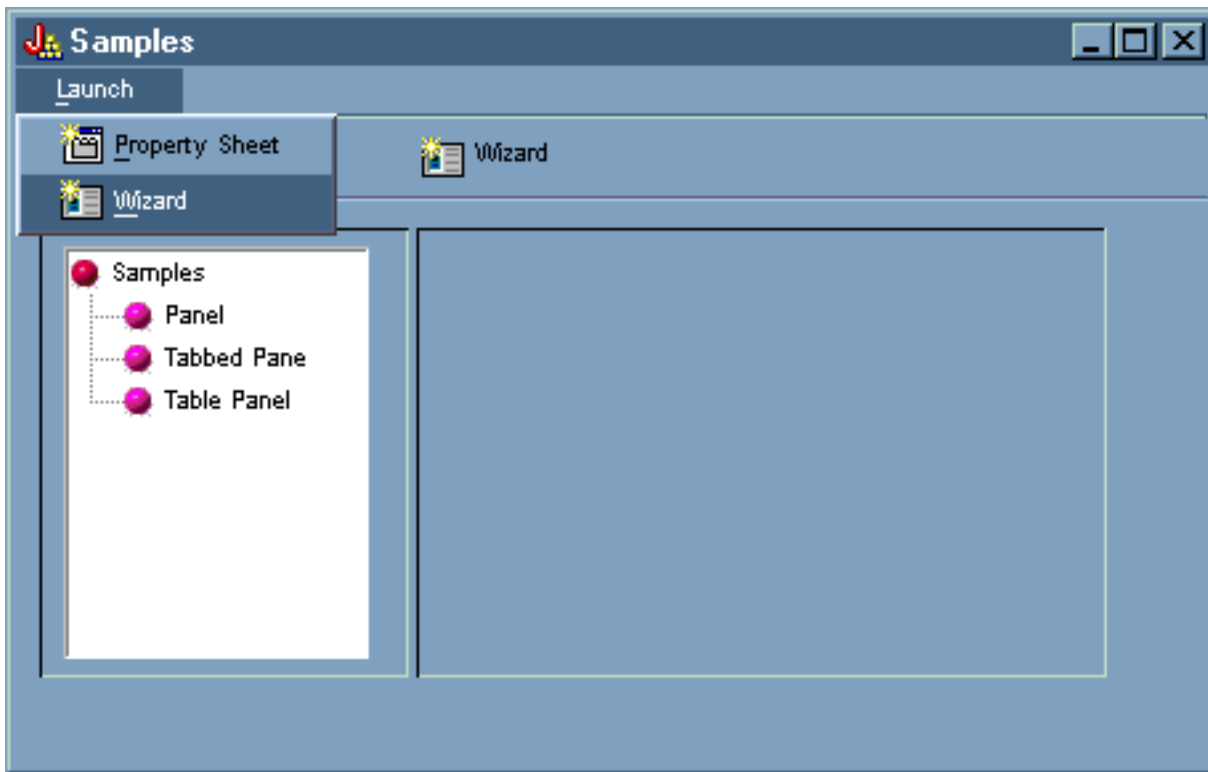
**Figura 7: selezione del separatore Editor cella tabella**



## Avviare il wizard

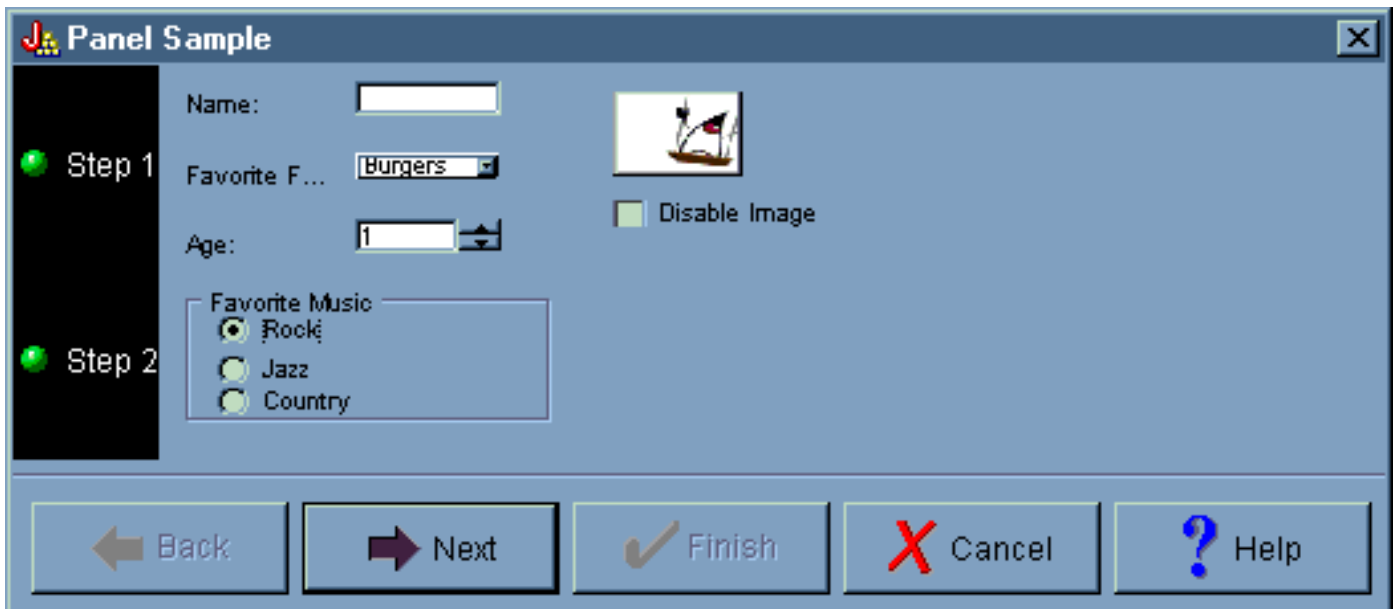
E' possibile avviare il wizard facendo clic sul pulsante Wizard della barra degli strumenti o utilizzando il menu **Avvia**. La possibilità di scegliere tra la barra degli strumenti e il menu consente di visualizzare collegamenti alle voci di menu. La figura 8 mostra il **Wizard** selezionato dal menu **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.

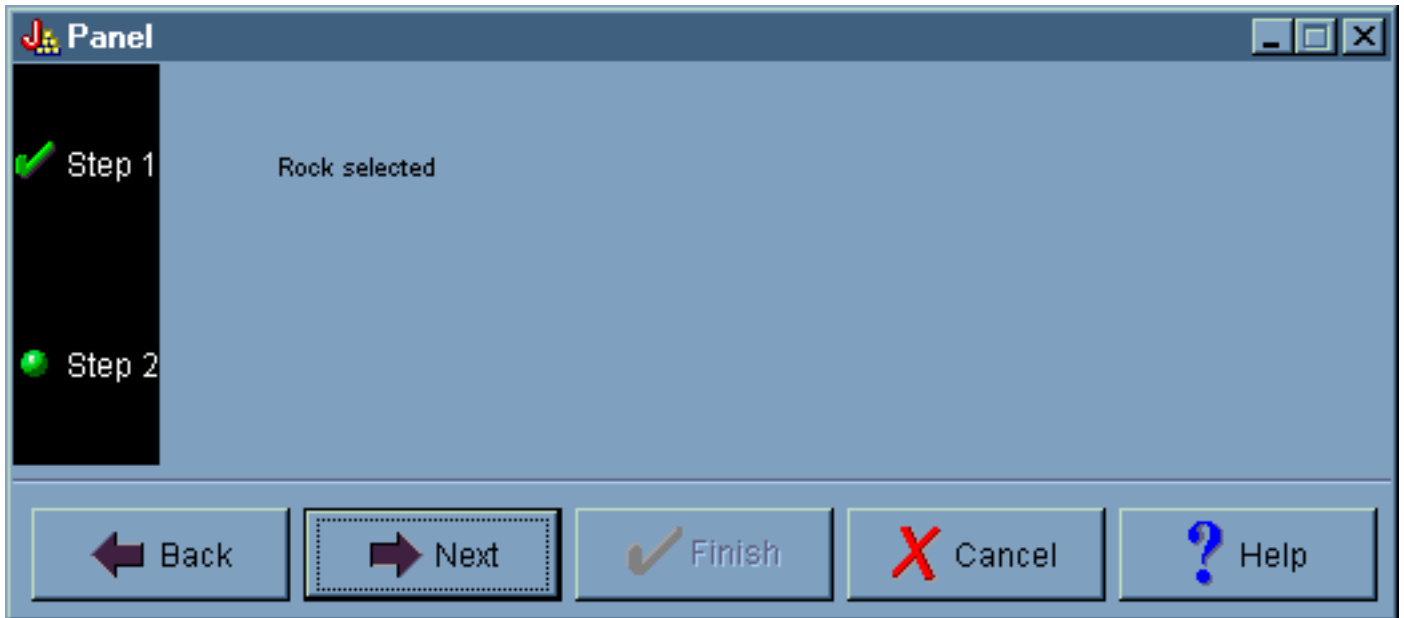
Figura 9: selezione di Rock nella prima finestra di dialogo del Wizard



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.

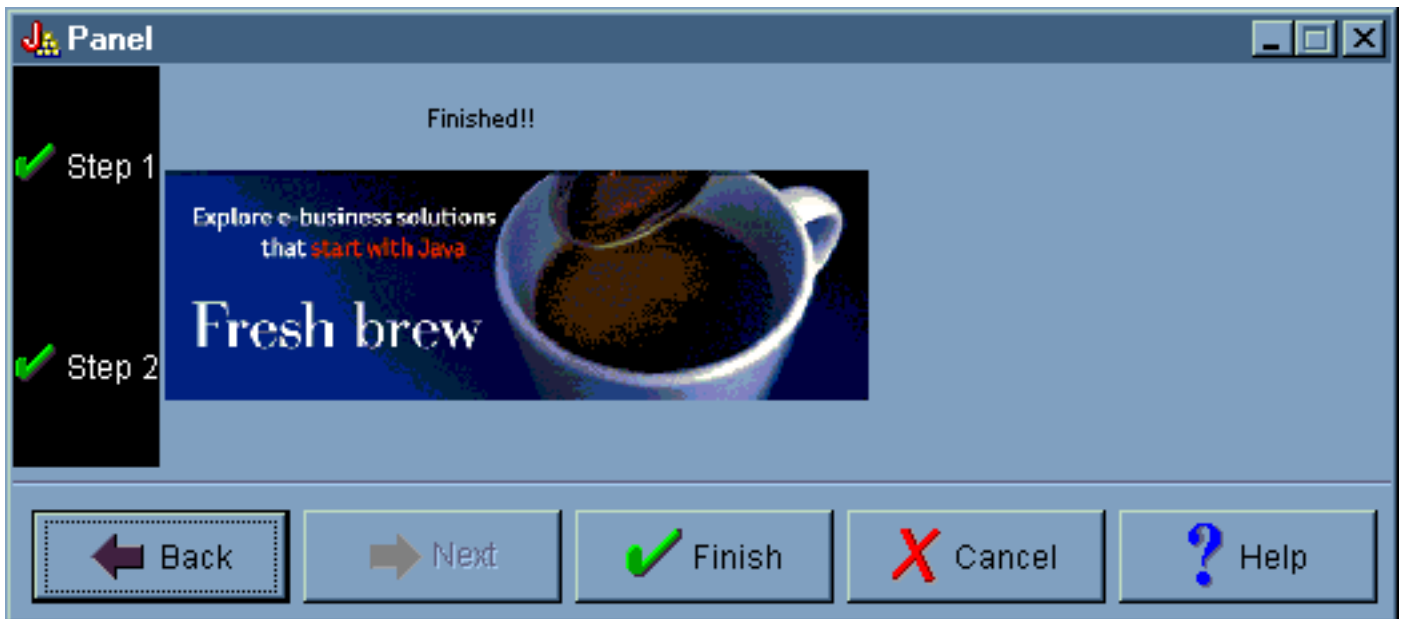
Figura 10: la seconda finestra di dialogo del wizard (dopo aver selezionato Rock)





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**

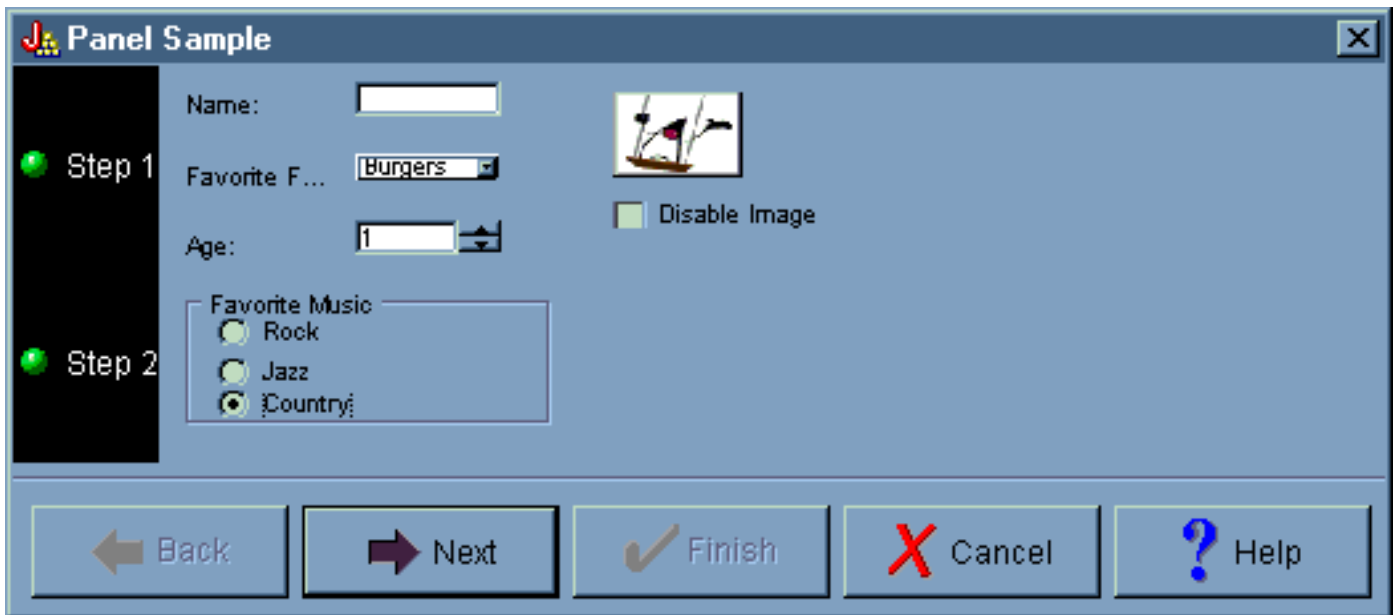


Figura 13: seconda finestra di dialogo del wizard (dopo aver selezionato Country)

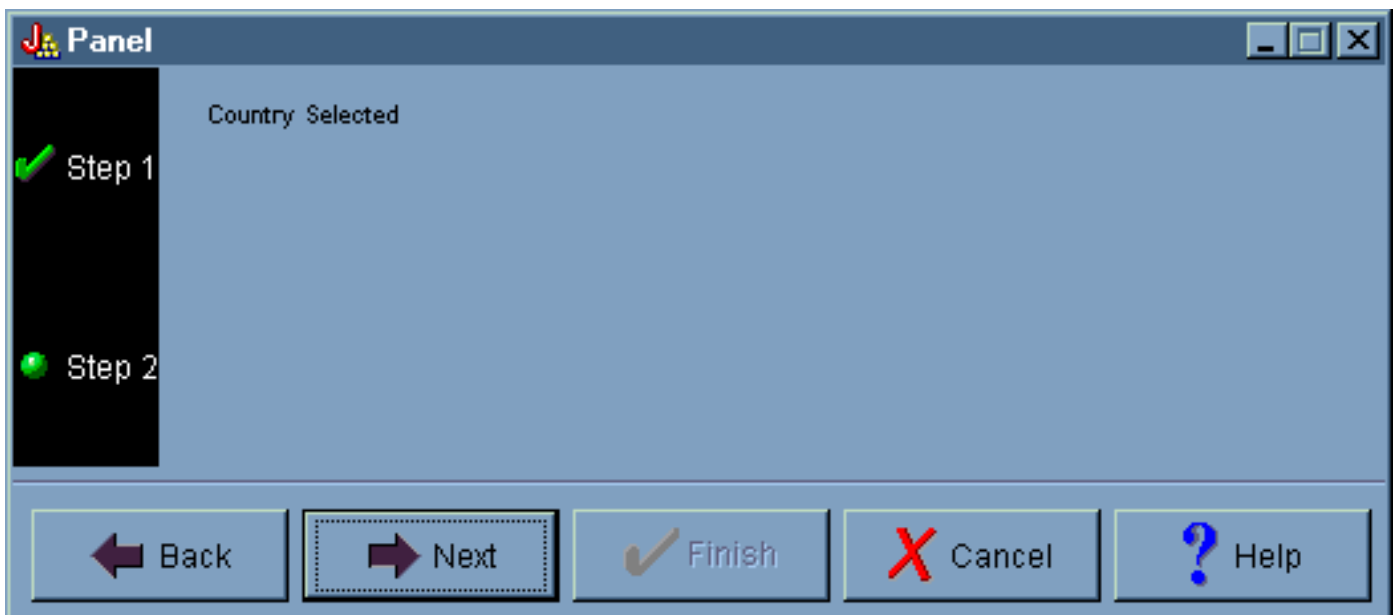
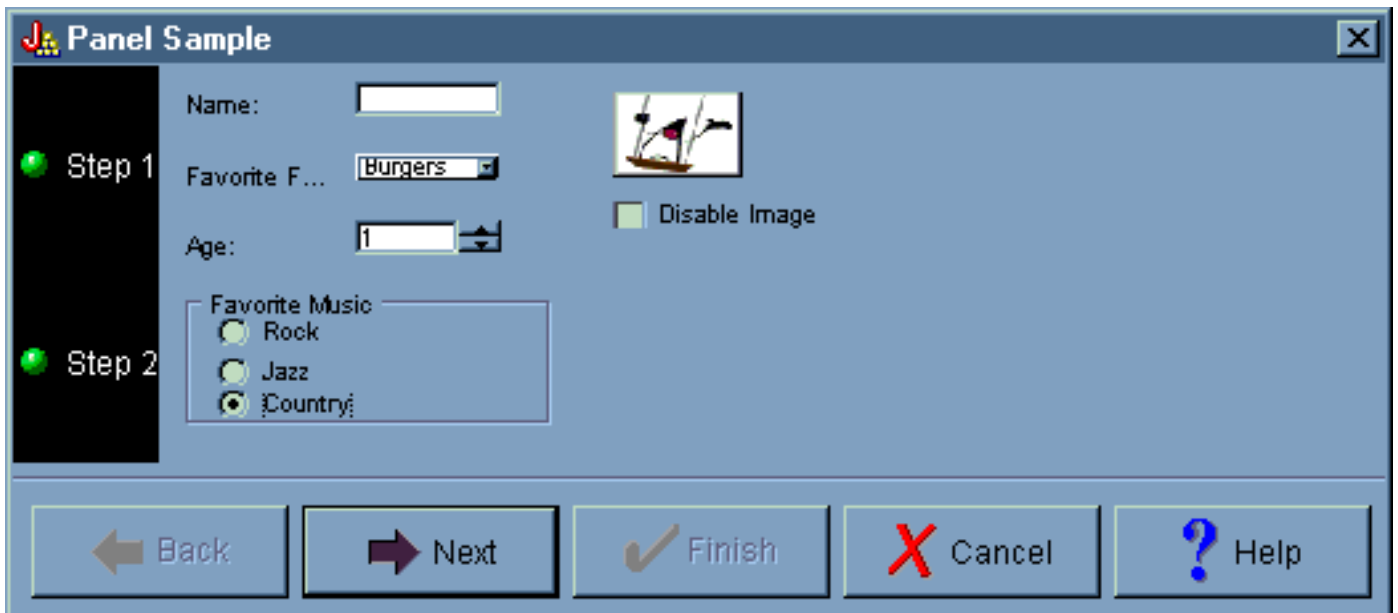


Figura 14: loop alla prima finestra di dialogo del wizard

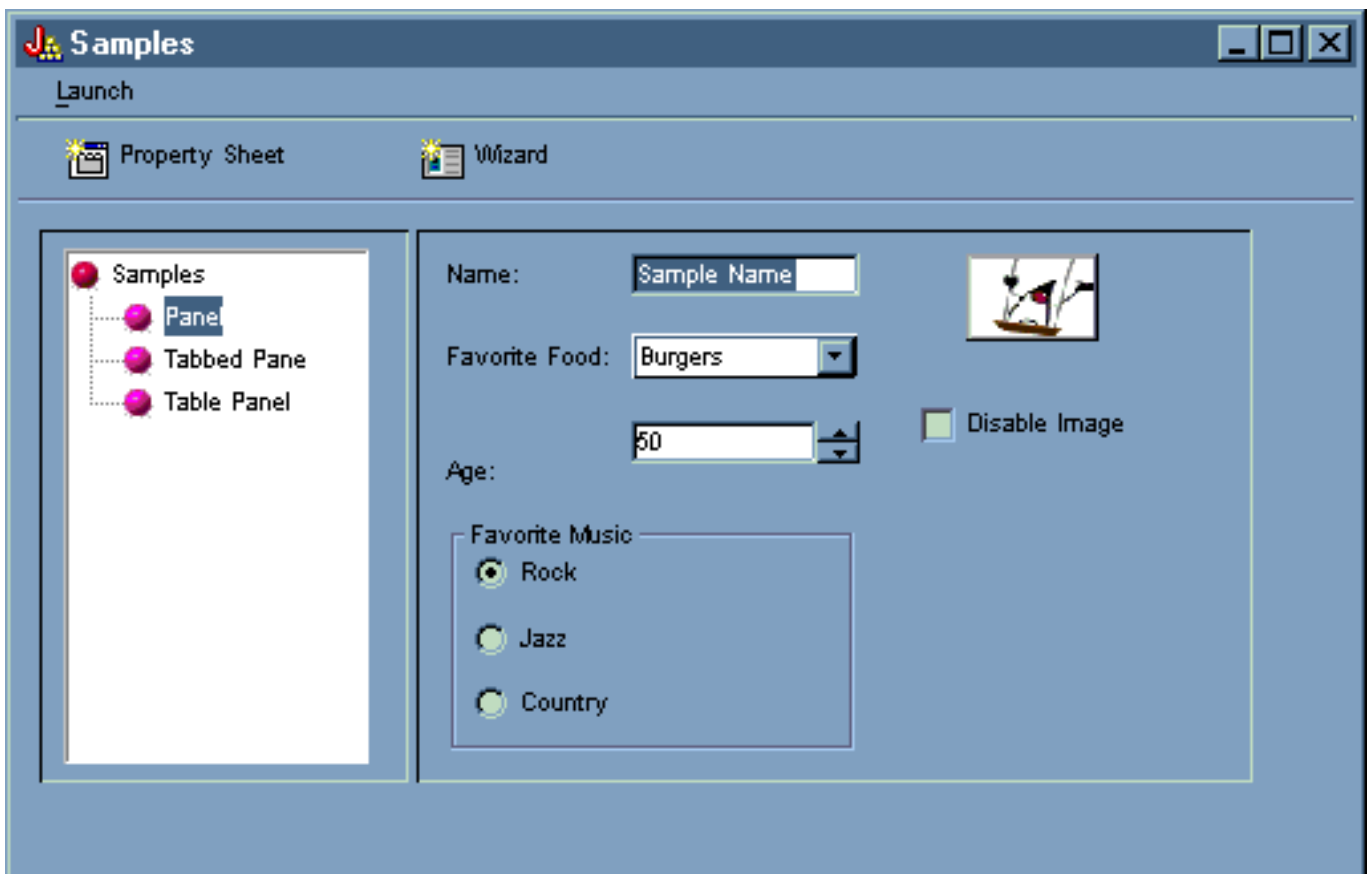


In altre parole, il programmatore ha stabilito che nessuno può selezionare Country come forma di musica preferita.

## Visualizzare i campioni

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.

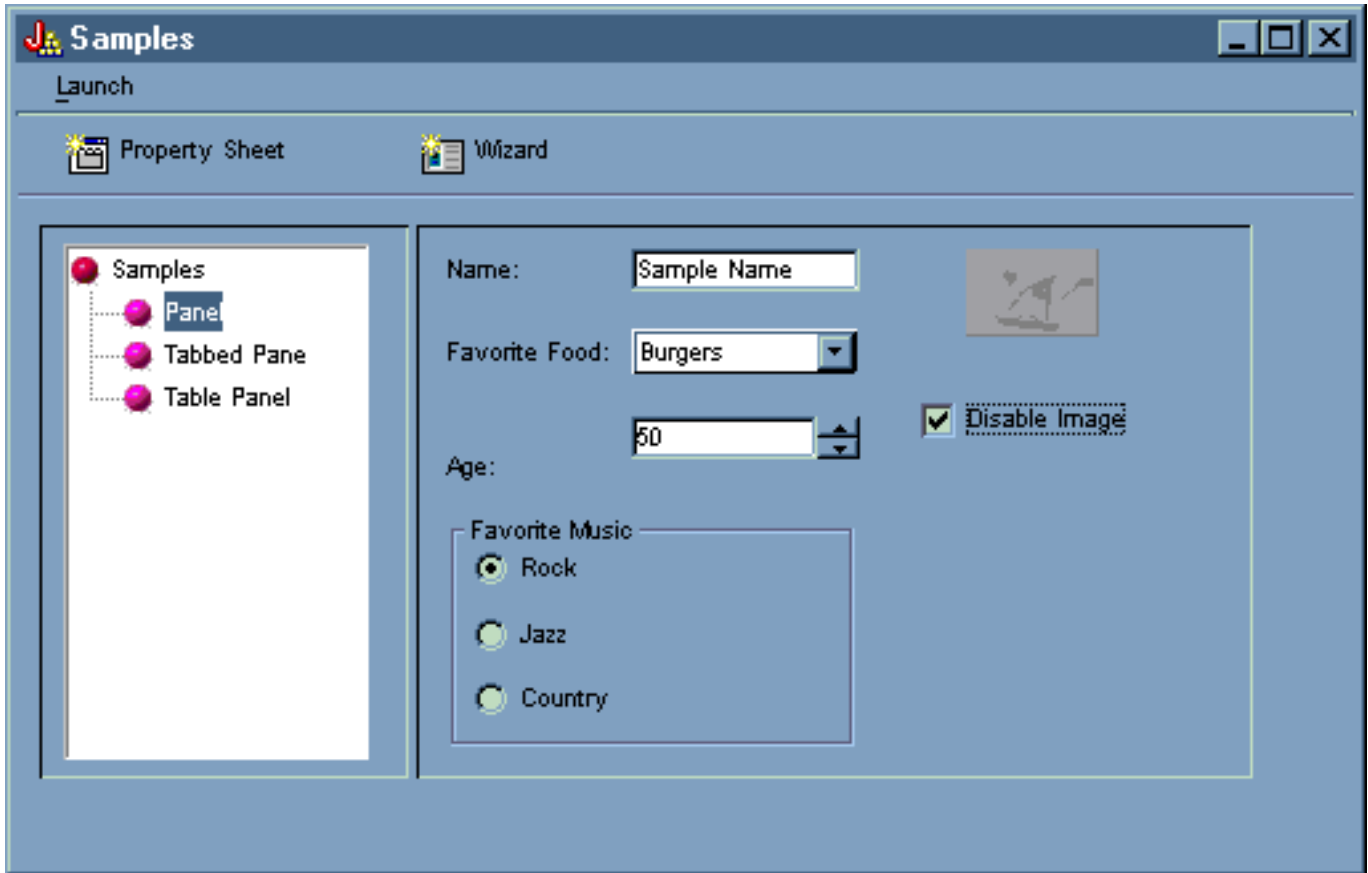
**Figura 15:** selezione di Pannello nel pannello di sinistra



Il Campione di pannello è stato programmato con un'opzione per disabilitare l'immagine. Selezionare **Disabilita**

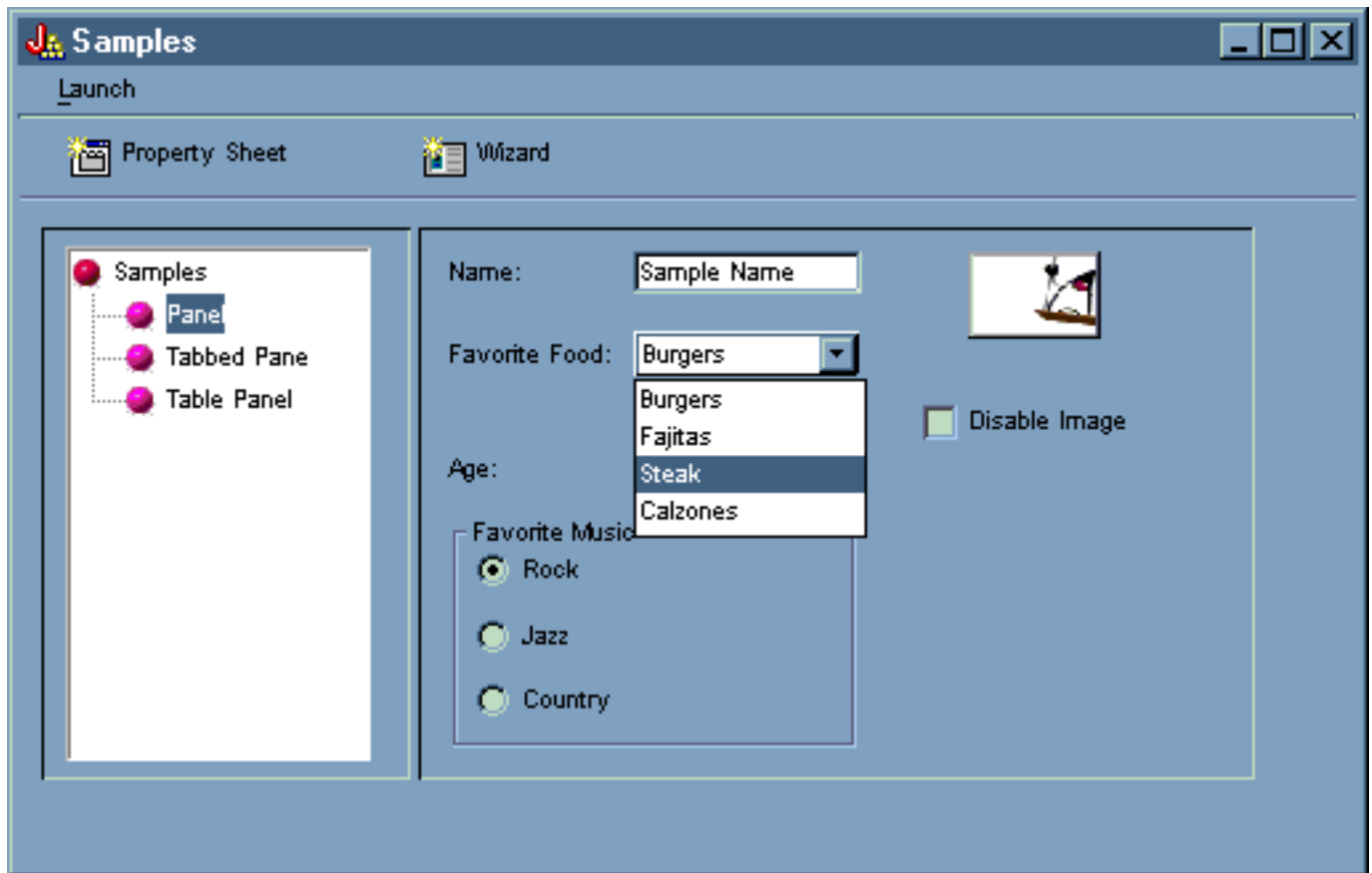
**immagine** per visualizzare lo stesso schermo con l'immagine ingrigita, come mostrato nella figura 16.

**Figura 16: selezione di Disabilita immagine nel pannello di destra**



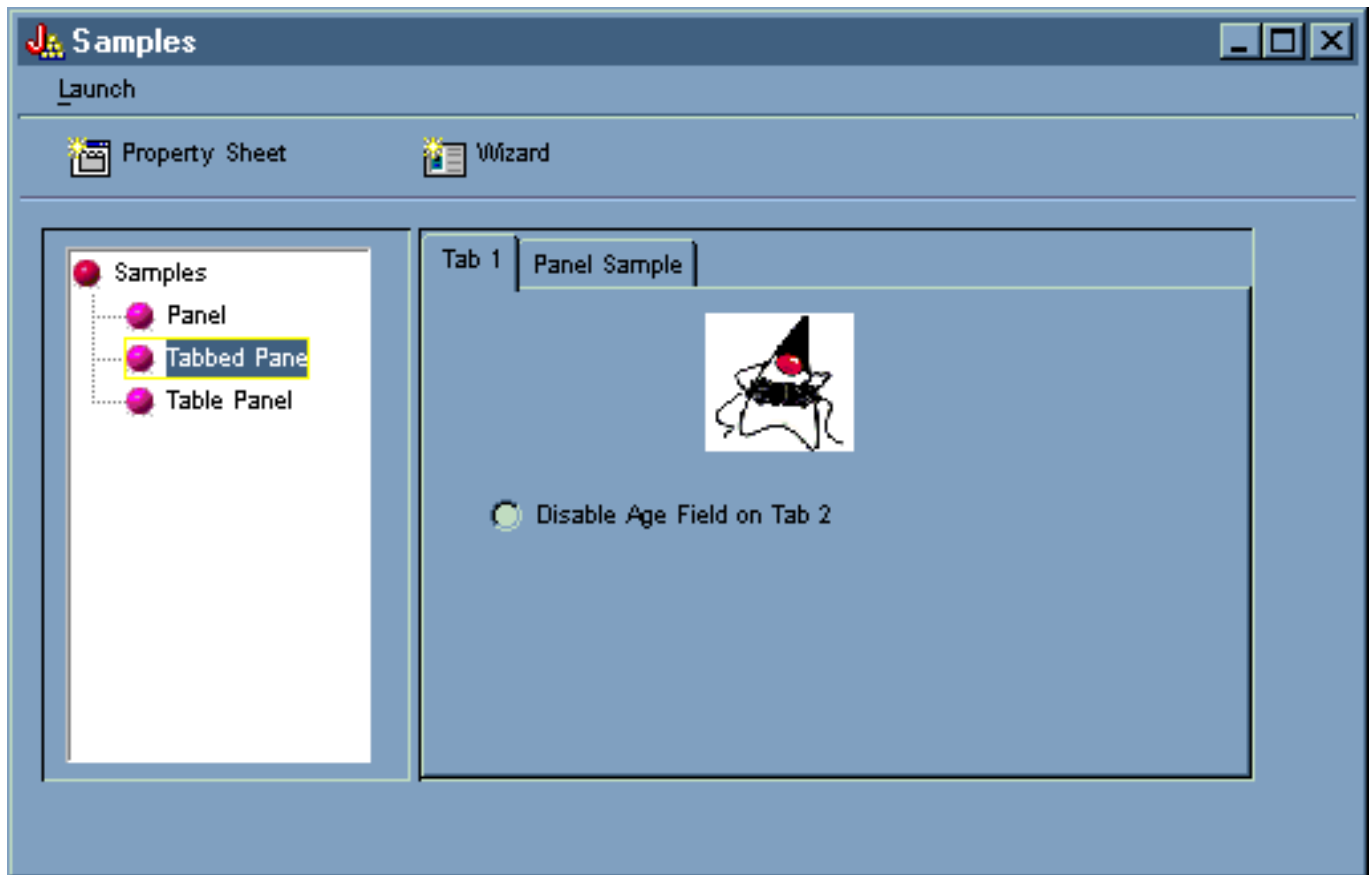
L'esempio di pannello illustra inoltre l'opzione della casella di elenco a discesa, come mostrato nella figura 17.

**Figura 17: selezione di un oggetto dalla lista Cibi preferiti nel pannello di destra**



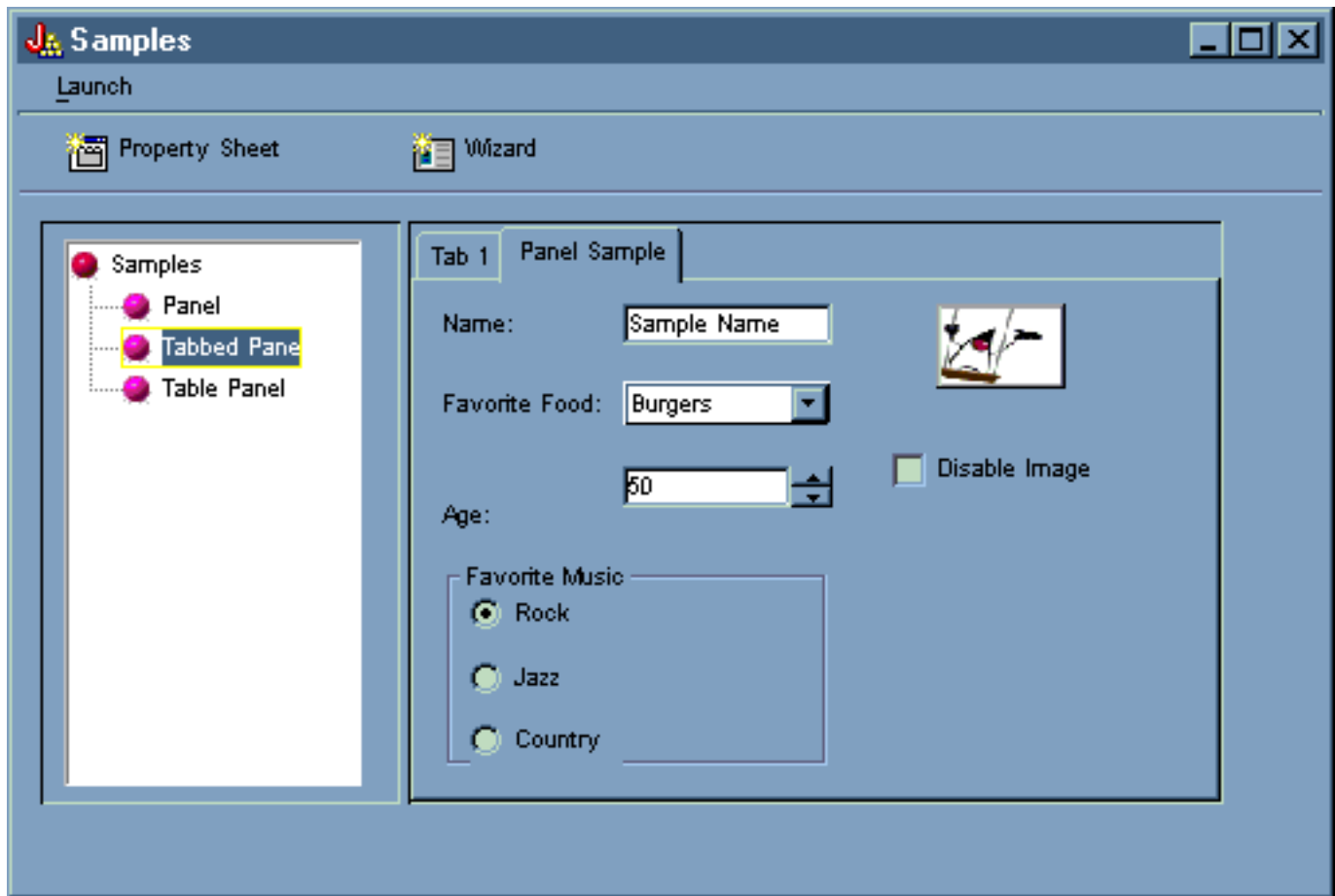
La figura 18 mostra come selezionando il **Pannello con separatori** nel pannello di sinistra della finestra principale dell'esempio del GUI Builder si visualizza il campione di Pannello con separatori nel pannello di destra.

**Figura 18: selezione del Pannello con separatori nel pannello di sinistra**



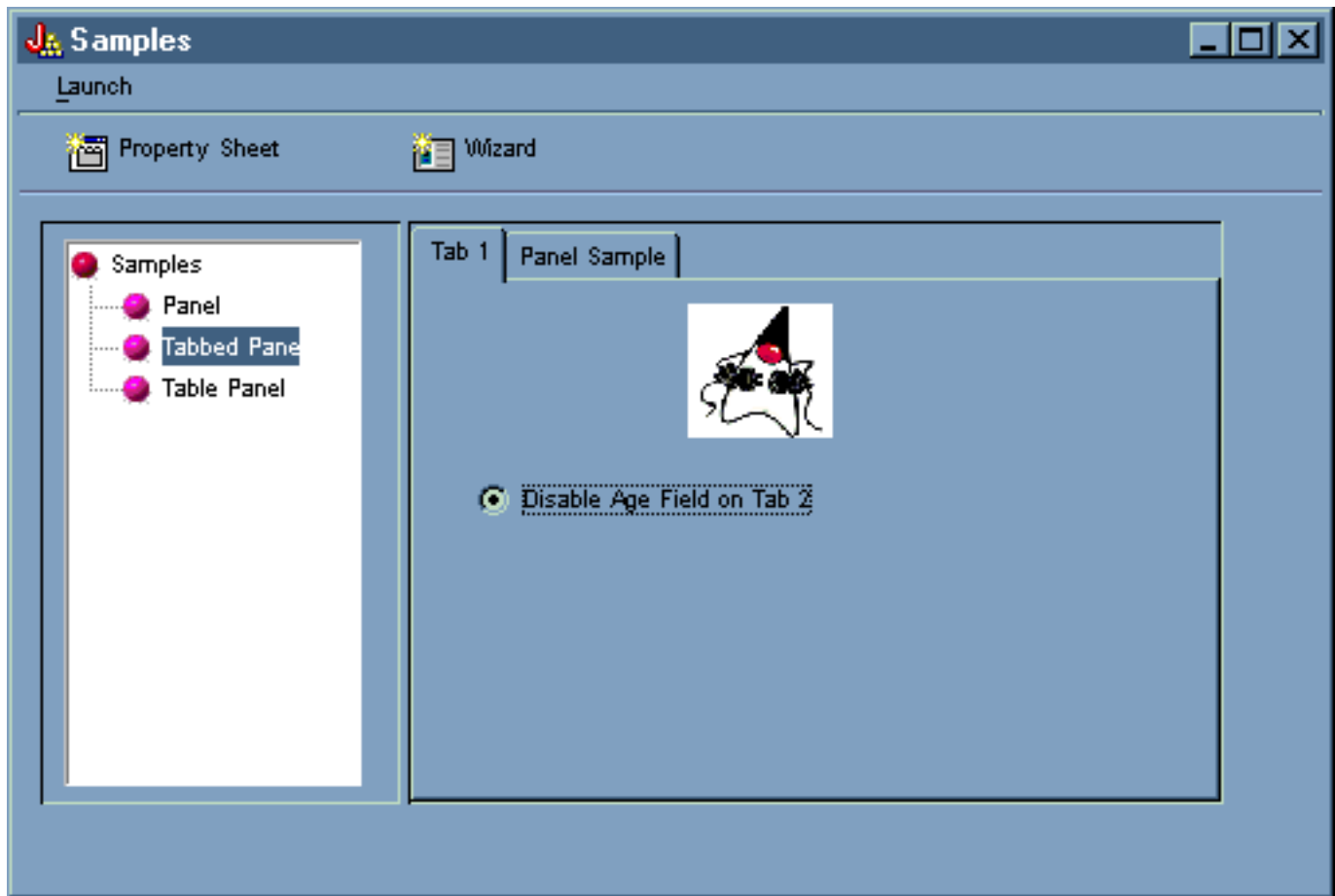
La figura 19 mostra i risultati della selezione del separatore **Campione del pannello** nel pannello di destra.

**Figura 19:** selezione del separatore **Campione del 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.

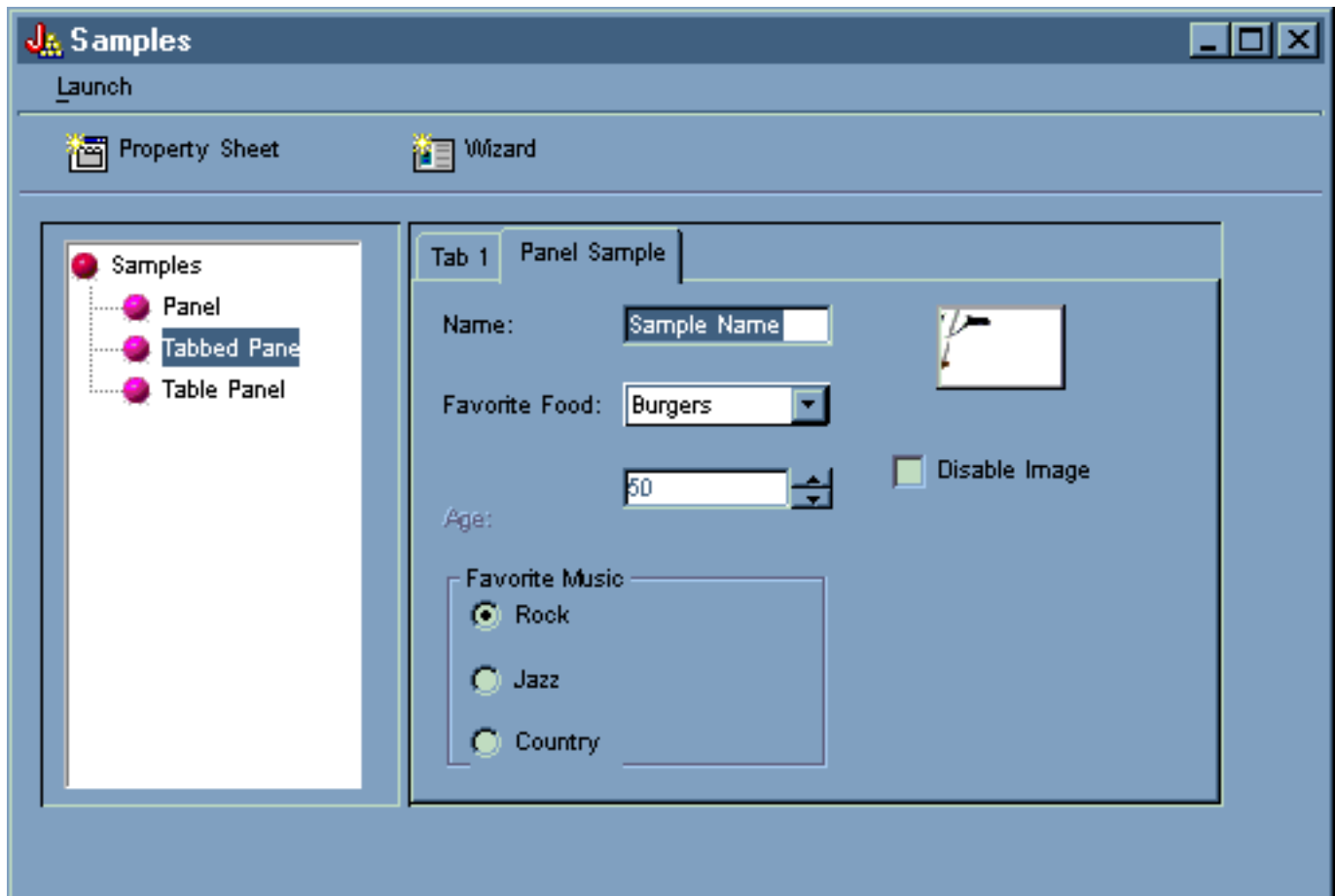
**Figura 20:** selezione di **Disabilita campo età sul Separatore 2** nel pannello di destra



La selezione dell'opzione **Disabilita campo età sul Separatore 2** disattiva e ingrigisce il campo **Età** nel separatore **Campione del pannello**, come mostrato nella figura 21.

**Figura 21: risultato della disabilitazione dell'età nel separatore Campione del 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**

Launch

Property Sheet

Wizard

- Samples
  - Panel
  - Tabbed Pane
  - Table Panel

Table with custom cell editor

Name	CheckBox Column
Sample Name	<input type="checkbox"/>

# Esempi dalle classi HTML

Gli esempi che seguono mostrano alcuni metodi necessari per utilizzare le classi HTML:

- [Esempio: utilizzare la classe BidiOrdering](#)
- [Esempio: creare oggetti HTMLAlign](#)
- [Esempio: utilizzare le classi HTMLForm](#)
- Esempi di classe di immissione modulo:
  - [Esempio: creare un oggetto ButtonFormInput](#)
  - [Esempio: creare un oggetto FileFormInput](#)
  - [Esempio: creare un oggetto HiddenFormInput](#)
  - [Esempio: creare un oggetto ImageFormInput](#)
  - [Esempio: creare un oggetto ResetFormInput](#)
  - [Esempio: creare un oggetto SubmitFormInput](#)
  - [Esempio: creare un oggetto TextFormInput](#)
  - [Esempio: creare un oggetto PasswordFormInput](#)
  - [Esempio: creare un oggetto RadioFormInput](#)
  - [Esempio: creare un oggetto CheckboxFormInput](#)
- [Esempio: creare oggetti HTMLHeading](#)
- [Esempio: utilizzare la classe HTMLHyperlink](#)
- [Esempio: utilizzare la classe HTMLImage](#)
- Esempi HTMLList
  - [Esempio: creare liste ordinate](#)
  - [Esempio: creare liste non ordinate](#)
  - [Esempio: creare liste nidificate](#)
- [Esempio: creare tag HTMLMeta](#)
- [Esempio: creare tag HTMLParameter](#)
- [Esempio: creare tag HTMLServlet](#)
- [Esempio: utilizzare la classe HTMLText](#)
- Esempi HTMLTree
  - [Esempio: utilizzare la classe HTMLTree](#)
  - [Esempio: creare una gerarchia ad albero IFS](#)
- Classi LayoutForm:
  - [Esempio: utilizzare la classe GridLayoutFormPanel](#)
  - [Esempio: utilizzare la classe LineLayoutFormPanel](#)
- [Esempio: utilizzare la classe TextAreaFormElement](#)
- [Esempio: utilizzare la classe LabelFormOutput](#)
- [Esempio: utilizzare la classe SelectFormElement](#)
- [Esempio: utilizzare la classe SelectOption](#)

- [Esempio: utilizzare la classe RadioFormInputGroup](#)
- [Esempio: utilizzare la classe RadioFormInput](#)
- [Esempio: utilizzare le classi HTMLTable](#)
  - [Esempio: utilizzare la classe HTMLTableCell](#)
  - [Esempio: utilizzare la classe HTMLTableRow](#)
  - [Esempio: utilizzare la classe HTMLTableHeader](#)
  - [Esempio: utilizzare la 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**

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: utilizzare le 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  
{  
  
    private static boolean found = false;           // Stabilire se l'utente  
    esiste già nella                               // lista degli utenti  
    iscritti.  
  
    String regPath = "c:\\registration.txt";       // Le informazioni sulla  
    registrazione verranno memorizzate in questa ubicazione  
  
    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 o l'e-mail degli utenti non vengono trovati 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();
    }
}

//-----

//-----

//Richiamo di "Numero dei server iSeries o AS/400e" 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 cliente informandolo che non sono
stati immessi il nome & e-mail del cliente. Quindi
        // è 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
LinearLayoutFormPanel line = new LinearLayoutFormPanel();
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 LinearLayoutFormPanel per il nome & l'indirizzo
e-mail
LinearLayoutFormPanel line2 = new LinearLayoutFormPanel();

// 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
LinearLayoutFormPanel line3 = new LinearLayoutFormPanel();

// Aggiungere elementi al layout di riga
line3.addElement(new LinearLayoutFormPanel());

```

```

        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 or AS/400e 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 alla lista 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);
//grid4.addElement(new LineLayoutFormPanel());
// 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 nel Modulo HTML nell'ordine di
visualizzazione desiderato
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 d 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();
}
}

```

# Esempio: utilizzare le 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);  
  
        // Toolbox utilizza una serie di icone predefinite per rappresentare  
        elementi espansi, compressi e documenti nell'HTMLTree.  
        // Per potenziare tali icone, Toolbox fornisce tre gif (expanded.gif,  
        collapsed.gif, bullet.gif) nel  
        // file jt400Servlet.jar. I browser non hanno la capacità di reperire  
        gif in un file jar o zip, quindi tali immagini  
        // devono essere estratte dal file jar e collocate nell'indirizzario  
        server web appropriato (per impostazione predefinita si tratta  
        // dell'indirizzario /html). Quindi eliminare il commento dalle  
        seguenti righe codice e specificare l'ubicazione corretta in questi  
        // metodi set. 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 la lista 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 nella lista.
            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: creare 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: creare 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 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 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");  
  
        // Il Toolbox utilizza una serie di icone predefinite per rappresentare  
        elementi espansi, compressi  
        // e documenti HTML nell'HTMLTree. Per potenziare tali icone, il  
        Toolbox fornisce
```

```

        // tre gif (expanded.gif, collapsed.gif, bullet.gif) nel file
jt400Servlet.jar.
        // I browser non hanno la capacità di individuare i gif in un file jar
o zip, quindi
        // è necessario estrarre quelle immagini dal file jar e collocarle
        // nell'indirizzario server web appropriato (per impostazione
predefinita è l'indirizzario /html). Quindi
        // modificare le righe di codice seguenti in modo da 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();
}

/**
 * This method will create the initial HTMLTree.
 */

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 la lista 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 nella lista.
                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: creare 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 informazioni legali importanti.

```
////////////////////////////////////  
//  
// Questo sorgente è un esempio di utilizzo delle classi pacchetto HTML di  
IBM Toolbox,  
// che consente di creare facilmente Liste 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 dalla
        // lista FileTreeElement nella frame di navigazione.
        if (req.getPathInfo() != null)
        {
            // Creare un FileListElement passando un oggetto di sistema
AS400 e la richiesta Http servlet.
            // 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());
        }
        else // Visualizzare questa HTMLHeading se non è stato
selezionato alcun FileTreeElement.
        {
            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: utilizzare le 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);

// Aggiungere righe alla tabella. Ogni 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.

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)*.
- [Richiamare una lista di informazioni](#): mostra il sorgente PCML ed il programma Java necessari per richiamare una lista di utenti autorizzati sul server. L'API che viene richiamata è l'API *Apertura lista utenti autorizzati (QGYOLAUS)*. Questo esempio illustra come accedere ad una schiera di strutture restituite da un programma del server.
- [Richiamare dati multidimensionali](#): mostra il sorgente PCML e il programma Java necessari per richiamare una lista 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 dell'autorizzazione a:

- 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

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">

<!-- PCML source for calling "Retrieve user Information" (QSYRUSRI) API -->

  <!-- Format USRI0150 - Other formats are available -->
  <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>

  <!-- Program QSYRUSRI and its parameter list for retrieving USRI0100
format -->
  <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" />
  </program>
</pcml>
```



```

    <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;

// Esempio di programma per richiamare l'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 il nome risorsa pcml. In questo esempio,
            // il file PCML serializzato "qsyrusri.pcml.ser" o
            // il file sorgente PCML "qsyrusri.pcml" deve trovarsi nel
classpath.
            pcml = new ProgramCallDocument(as400System, "qsyrusri");

```

```

    // Impostare i parametri di immissione. Diversi parametri hanno valori
predefiniti
    // 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 la lista dei 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: richiamare una lista 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">

<!-- PCML source for calling "Open List of Authorized Users" (QGYOLAUS) API
-->

<!-- Format AUTU0150 - Other formats are available -->
<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>

<!-- List information structure (common for "Open List" type APIs) -->
<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>

<!-- Program QGYOLAUS and its parameter list for retrieving AUTU0150
format -->
<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>

<!-- Program QGYGTLE returned additional "records" from the list
      created by 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>

<!-- Program QGYCLST closes the list, freeing resources on the 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 per 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 della lista 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 il nome risorsa pcml. In questo esempio,
        // il file PCML serializzato "qgyolaus.pcml.ser" o
        // il file sorgente 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 la lista dei 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 la lista 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 negativo, sarà necessario effettuare chiamate a
"Richiamo voci lista" (QGYGTLE)
        // successivamente per richiamare gli utenti rimanenti nella
lista.
        listStatus = (String) pcml.getValue(programName +
".listInfo.listStatus");
        if ( listStatus.equals("2") // La lista è contrassegnata come
"Completa"
|| listStatus.equals("3") ) // Oppure è contrassegnata con
"Errore creazione"
        {
            doneProcessingList = true;
        }
        else
        {
            programName = "qgygtle";

            // Impostare i valori di immissione per QGYGTLE
            pcml.setValue("qgygtle.requestHandle", requestHandle);
            pcml.setIntValue("qgygtle.startingRcd", nbrUsers + 1);

            // Chiamare "Richiamo voci lista" (QGYGTLE) per richiamare altri
utenti dalla lista
            rc = pcml.callProgram("qgygtle");

            // Se il codice di ritorno è false, si riceveranno messaggi dal server
            if(rc == false)
            {
                // Reperire la lista dei 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);

// Chiamare l'API "Chiusura lista" (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: richiamare 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;

// Esempio di programma per chiamare 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 il nome risorsa pcml. In questo esempio,
            // il file PCML serializzato "qznfrtve.pcml.ser" o
            // il file sorgente PCML "qznfrtve.pcml" devono trovarsi nel
classpath.
            pcml = new ProgramCallDocument(as400System, "qznfrtve");

            // Impostare i parametri di immissione. Diversi parametri hanno valori
predefiniti
            // 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("qznfirtve");

if (rc == false)
{
    // Reperire la lista dei messaggi server
    AS400Message[] msgs = pcml.getMessageList("qznfirtve");

    // 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("qznfirtve.returnedRcdsFdbkInfo.nbrOfNFSExportEntries");
        // Iterare la lista delle esportazioni
        for (indices[0] = 0; indices[0] < nbrExports; indices[0]++)
        {
            value = pcml.getValue("qznfirtve.receiver.pathName", indices);
            System.out.println("Path name = " + value);

            // Iterare e scrivere Read Write Host Names per questa
esportazione
            nbrOfReadWriteHostNames =
pcml.getIntValue("qznfirtve.receiver.nbrOfReadWriteHostNames", indices);
            for(indices[1] = 0; indices[1] < nbrOfReadWriteHostNames;
indices[1]++)
            {
                value = pcml.getValue("qznfirtve.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("qznfirtve.receiver.nbrOfRootHostNames", indices);
            for(indices[1] = 0; indices[1] < nbrOfRootHostNames; indices[1]++)
            {
                value =
pcml.getValue("qznfirtve.receiver.rootAccessList.hostName", indices);
                System.out.println("    Root access host name = " + value);
            }

            // Iterare e scrivere Access Host Names per questa esportazione
            nbrOfAccessHostnames =

```

```

pcml.getIntValue("qznfirtve.receiver.nbrOfAccessHostNames", indices);
    for(indices[1] = 0; indices[1] < nbrOfAccessHostnames;
indices[1]++)
    {
        value =
pcml.getValue("qznfirtve.receiver.accessHostNames.hostName", indices);
        System.out.println("    Access host name = " + value);
    }

    // Iterare e scrivere Host Options per questa esportazione
    nbrOfHostOpts =
pcml.getIntValue("qznfirtve.receiver.nbrOfHostOptions", indices);
    for(indices[1] = 0; indices[1] < nbrOfHostOpts; indices[1]++)
    {
        System.out.println("    Host options:");
        value =
pcml.getValue("qznfirtve.receiver.hostOptions.dataFileCodepage", indices);
        System.out.println("        Data file code page = " + value);
        value =
pcml.getValue("qznfirtve.receiver.hostOptions.pathNameCodepage", indices);
        System.out.println("        Path name code page = " + value);
        value =
pcml.getValue("qznfirtve.receiver.hostOptions.writeModeFlag", indices);
        System.out.println("        Write mode flag = " + value);
        value = pcml.getValue("qznfirtve.receiver.hostOptions.hostName",
indices);
        System.out.println("        Host name = " + value);
    }
    } // end for loop iterating list of exports
} // end call to QZNFRTVE succeeded
}
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 codici che sono forniti in tutta la documentazione delle classi ReportWriter.

## JSPReportProcessor e PDFContext

- [Esempio: utilizzare JSPReportProcessor con PDFContext](#)
- [»Esempio: file JSP di esempio JSPReportProcessor«](#)

## XSLReportProcessor e PCLContext

- [Esempio: utilizzare XSLReportProcessor con PCLContext](#)
- [»Esempio: file XSL di esempio di ReportProcessor«](#)
- [»Esempio: file XSL 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**

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: utilizzare JSPReportProcessor con PDFContext

**Nota:** leggere l'[Esonero di responsabilità per gli esempi di codice](#) per informazioni legali importanti.

```
////////////////////////////////////
//
// Il seguente esempio (JSPRunReport) utilizza le classi JSPReportProcessor
// e
// PDFContext per ottenere dati da un URL specificato e convertire i dati
// nel formato PDF. Il flusso di dati viene quindi inviato ad un file come
// documento PDF.
//
// Per visualizzare il contenuto di un file sorgente JSP di esempio che si
// può utilizzare con
// JSPRunReport, consultare JSPcust\_table.jsp. E' possibile inoltre
// scaricare un file zip
// che contiene un file di esempio JSP. Il file zip contiene anche file di
// esempio
// XML e XSL che è possibile utilizzare con l'esempio di XSLReportProcessor
// (PCLRunReport).
//
// 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 l'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 un OutputStream **/
PDFContext pdfcontext = new PDFContext((OutputStream)fileout, pf);

System.out.println( Ready to parse XSL document );

/** creare l'oggetto JSPReportProcessor ed impostare la mascherina
    sul 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;
  }
%>

<!-- First test of parse and compose. -->
<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:root>
```

```

</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: utilizzare 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. Si può
anche
// scaricare un file zip che contiene i file di esempio XML ed 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 system 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 informazioni legali importanti.

```
<?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>
</RESIDENTIAL-LISTING>
</RESIDENTIAL-LISTINGS>
```

</FEATURES>

<FINANCIAL>

<ASSUMABLE>

I assume so.

</ASSUMABLE>

<OWNER-CARRY>

Too heavy.

</OWNER-CARRY>

<ASSESSMENTS>

\$150,000

</ASSESSMENTS>

<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>
    </GENERAL>
</RESIDENTIAL-LISTING>

```

```
<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>
```

<ASSESSMENTS>  
\$150,000  
</ASSESSMENTS>  
<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>
  <ASSESSMENTS>
    $150,000
  </ASSESSMENTS>
  <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>
  <ASSESSMENTS>
    $300,000
  </ASSESSMENTS>
  <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 informazioni legali importanti.

```
<?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 di risorse.

## Resource e ChangeableResource

- [Esempio: richiamare un valore di attributo da RUser](#), una reale sottoclasse Resource
- [Esempio: impostare i valori di attributo per RJob](#), una reale sottoclasse di ChangeableResource
- [Esempio: utilizzare un codice generico per accedere alle risorse](#)

## ResourceList

- [Esempio: richiamare e stampare il contenuto di una ResourceList](#)
- [Esempio: utilizzare un codice generico per accedere a una ResourceList](#)
- [Esempio: visualizzare una lista di risorse in un servlet](#)

## Presentazione

- [Esempio: utilizzare le 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.

# Esempio: richiamare 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: impostare i 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: utilizzare un codice generico per accedere alle risorse

E' possibile scrivere un codice generico per gestire qualsiasi sottoclasse Resource o ChangeableResource. Tale codice può migliorare la capacità di riutilizzo e di manutenzione e gestirà le future sottoclassi Resource o ChangeableResource senza necessitare di alcuna modifica.

Ogni attributo ha un oggetto metadati dell'attributo associato ([com.ibm.as400.resource.ResourceMetaData](http://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.

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);
    }
}
```

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: lista di risorse

I seguenti esempi mostrano le varie modalità di gestione delle liste delle risorse:

- Esempio: [richiamare e stampare il contenuto di una ResourceList](#)
- Esempio: [utilizzare un codice generico per accedere ad una ResourceList](#)
- Esempio: [visualizzare una lista di risorse in un servlet](#)

## Esempio: richiamare e stampare il contenuto di una ResourceList

Un esempio di una sottoclasse concreta di ResourceList è [com.ibm.as400.resource.RJobList](#), che rappresenta una lista di lavori iSeries. RJobList supporta molti [ID selezione](#) e [ID ordinamento](#), ognuno dei quali può essere utilizzato per filtrare o ordinare la lista. Questo è un esempio per stampare il contenuto di una RJobList:

```
// Creare un oggetto RJobList per rappresentare una lista di lavori.
AS400 system = new AS400("MYSYSTEM", "MYUSERID", "MYPASSWORD");
RJobList jobList = new RJobList(system);

// Filtrare la lista in modo da includere solo i lavori interattivi.
jobList.setSelectionValue(RJobList.JOB_TYPE, RJob.JOB_TYPE_INTERACTIVE);

// Ordinare la lista per nome utente, quindi per nome lavoro.
Object[] sortValue = new Object[] { RJob.USER_NAME, RJob.JOB_NAME };
jobList.setSortValue(sortValue);

// Aprire la lista e attenderne il completamento.
jobList.open();
jobList.waitForComplete();

// Leggere e stampare il contenuto della lista.
long length = jobList.getListLength();
for(long i = 0; i < length; ++i)
{
    System.out.println(jobList.resourceAt(i));
}

// Chiudere la lista.
jobList.close();
```

## Esempio: utilizzare un codice generico per gestire le risorse.

Oltre ad utilizzare direttamente le sottoclassi concrete, è possibile scrivere un codice generico per gestire ogni sottoclasse di ResourceList. Tale codice può migliorare la capacità di riutilizzo e di manutenzione e consentirà di gestire le future sottoclassi ResourceList senza richiedere alcuna modifica.

### Esempio: stampare il contenuto di una 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 la lista 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: utilizzare ResourceMetaData per accedere ad ogni attributo supportato da una risorsa**

Ogni attributo ha un oggetto metadati dell'attributo associato ([com.ibm.as400.resource.ResourceMetaData](http://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.

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: utilizzare ResourceMetaData per ripristinare 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();
}

```



## Esempio: presentare una lista di risorse in un servlet

Utilizzare la classe `ResourceListRowData` insieme alla classe `HTMLFormConverter` o `HTMLTableConverter` per presentare una lista di risorse in un servlet.

- `HTMLFormConverter` visualizza il contenuto di una lista di risorse come una serie di moduli, in cui ogni modulo contiene i valori dell'attributo per una risorsa della lista.
- `HTMLTableConverter` visualizza il contenuto di una lista di risorse come una tabella, in cui ogni riga contiene le informazioni riguardo una risorsa della lista.

Le colonne per un oggetto `ResourceListRowData` vengono specificate come una schiera di ID dell'attributo colonna mentre ogni riga rappresenta un oggetto risorsa.

```
// Creare la lista risorse. Questo esempio crea una lista di
// tutti i messaggi presentinella 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]);
```

## »Esempi: RFML

Questa sezione elenca gli esempi di codice che sono forniti in tutta la documentazione RFML:

- [Esempio: utilizzare l'RFML rispetto all'utilizzo delle classi Record 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**

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 "COSÌ COME SONO" senza garanzie di alcun tipo. Le garanzie assicurate di non violazione, commerciabilità ed idoneità ad un particolare scopo sono smentite espressamente. «

# Esempio: utilizzare 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 informazioni legali importanti.

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.setToken("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: utilizzare la classe ListRowData](#)
- [Esempio: utilizzare la classe RecordListRowData](#)
- [Esempio: utilizzare la classe ResultSetRowData](#)
- [Esempio: utilizzare la classe HTMLFormConverter](#)
- [Esempio: utilizzare la classe ListMetaData](#)
- [Esempio: utilizzare la classe ResultSetMetaData](#)
- [Esempio: presentare una lista risorse in un servlet](#)

E' inoltre possibile utilizzare il servlet e le classi [HTML](#) 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**

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: utilizzare ListRowData

Questo esempio è composto da tre parti:

- [Sorgente Java](#) che illustra il funzionamento della classe ListRowData
- [Sorgente HTML](#) generato dal sorgente Java tramite [HTMLTableConverter](#)
- [Visualizzazione dell'HTML creato da parte del browser](#)

## 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, RowMetaData.Type.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, RowMetaData.Type.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 della classe [HTMLTableConverter](#) nell'esempio di sorgente Java precedentemente riportato 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.

<b>ID cliente</b>	<b>Numero di ordine</b>
777-53-4444	12345-XYZ
777-53-4444	56789-ABC

# Esempio: utilizzare 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 come funziona la 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 creato 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>CUSNUM</th>
<th>LSTNAM</th>
<th>INIT</th>
<th>STREET</th>
<th>CITY</th>
<th>STATE</th>
<th>ZIPCOD</th>
<th>CDTLMT</th>
<th>CHGCOD</th>
<th>BALDUE</th>
<th>CDTDUE</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>
</table>
```

## Come un browser visualizza l'HTML creato

La seguente tabella mostra come appare il codice sorgente HTML quando viene visualizzato in un browser.

CUSNUM	LSTNAM	INIT	STREET	CITY	STATE	ZIPCOD	CDTLMT	CHGCOD	BALDUE	CDTDUE
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: utilizzare ResultSetRowData

Questo esempio è composto da tre parti:

- [Sorgente Java](#) che mostra come funziona la classe ResultSetRowData
- [Sorgente HTML](#) generato dal sorgente Java tramite [HTMLTableConverter](#)
- [Modalità di visualizzazione dell'HTML creato da parte del browser](#)

## Il sorgente Java che mostra il funzionamento della classe ResultSetRowData

```
// 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 ResultSetRowData e inizializzarlo nella
serie di risultati.
ResultSetRowData rowData = new ResultSetRowData(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",
"State", "Zip Code",
"Street Address", "City",
"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 creato 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>
```

## Come un browser visualizza l'HTML creato

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
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: utilizzare 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.ResultSetRowData;

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 a principale", tornare al
modulo HTML principale
else if (parameters.containsKey("returnToMain"))
{
    session.invalidate();
    cleanup();
    out.println(showHtmlMain());
}
// se è stato premuto il pulsante "Primo", visualizzare il primo
record
else if (parameters.containsKey("getFirstRecord"))
{
    rowData.first();
    out.println(showHtmlForRecord(htmlTable, 0));
}
// se è stato premuto il pulsante "Precedente", visualizzare il record
precedente
else if (parameters.containsKey("getPreviousRecord"))
{
    if (!rowData.previous())
    {
        rowData.first();
    }
    out.println(showHtmlForRecord(htmlTable,
rowData.getCurrentPosition()));
}

```

```

    }
    // se è stato premuto il pulsante "Successivo", visualizzare 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", visualizzare 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 ResultSetRowData getAllRecords(Hashtable parameters,
ServletOutputStream out)
throws IOException
{
    ResultSetRowData records = null;

    try
    {
        // Richiamare il nome sistema, libreria e file dalla lista
    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 il ResultSet
                // o potrebbero verificarsi 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 ed 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 la lista 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 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 " +
server " +
array of HTML " +
record from " +
" +
backward " +
"to convert data retrieved from a
file. The converter produces an
tables. Each entry in the array is a
file. " +
"Records are displayed one at a time,
"giving you buttons to move forward or
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, " +
the file you " +
Records " +
"and the file and library name for
wish to access. Then push the Show
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><!-- Output from the HTMLFormConverter
class-->
</b></font></td>
</tr>
</table>
</td>
</tr>
<tr>
<td><form>
<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.*;

/*
An example of using Toolbox classes in a servlet.

Schemas of SQL databases on the 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 stia inoltrando le informazioni. Prendere le informazioni 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 una lista delle 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 aggiungeli.
        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 una lista 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 una lista 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 genererà 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 genererà 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();
}
}

```

## » Creare il primo programma 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 si trova nell'indirizzario c:\lib sulla stazione di lavoro (su cui è in esecuzione Windows), aggiungere quanto segue alla fine dell'istruzione CLASSPATH:

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

3. Aprire un editor di testo e inserire l'[il primo semplice esempio di programmazione](#).

**Nota:** accertarsi di escludere il testo che fa riferimento 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 e utilizzare il comando seguente per compilare il semplice esempio di programmazione:

```
javac CmdCall.java
```

5. Nella sessione comandi, immettere il comando seguente per eseguire il semplice esempio di programmazione:

```
java CmdCall
```






Per visualizzare informazioni dettagliate in una finestra separata, il browser deve avere il supporto JavaScript abilitato. Se il browser non ha il JavaScript abilitato o non supporta JavaScript, fare clic sul collegamento testo **Nota** o andare alla fine della pagina per visualizzare le informazioni.

[ [Esempi semplici di programmazione](#) ]

## Esempio: utilizzare CommandCall

Utilizzare quanto segue come esempio per il proprio programma. L'esempio include informazioni dettagliate sulle righe chiave nel codice. E' possibile visualizzare le informazioni dettagliate nei seguenti modi:

- Fare clic sull'immagine  per visualizzare le informazioni dettagliate in una finestra a comparsa.
- Fare clic sul collegamento testo per visualizzare le informazioni dettagliate fornite alla fine dell'esempio.

**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 Toolbox si trovano in com.ibm.as400.access.package.  
// Importare questo pacchetto per utilizzare le classi Toolbox.  
//  
////////////////////////////////////  
  
import com.ibm.as400.access.*;  
  
public class CmdCall  
{  
    public static void main (String[] args)  
    {  
        // Come altre classi Java le classi Toolbox emettono  
        // eccezioni quando si presenta qualche problema. Tali eccezioni  
        devono essere  
        // catturate dai programmi che utilizzano Toolbox.  
        tryNote_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. Toolbox per Java utilizza l'oggetto "AS400" per identificare il server di destinazione. Se si crea l'oggetto AS400 senza parametri, Toolbox per Java richiede il nome di 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 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 metodo run() sull'oggetto command call per eseguire un comando.
4. Il risultato dell'esecuzione di un comando è una lista di messaggi OS/400. Il Toolbox rappresenta questi 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.


[ [Esempi semplici di programmazione](#) ]

Per visualizzare informazioni dettagliate in una finestra separata, il browser deve avere il supporto JavaScript abilitato. Se il browser non ha il JavaScript abilitato o non supporta JavaScript, fare clic sul collegamento testo **Nota** o andare alla fine della pagina per visualizzare le informazioni.

[ [Parte successiva](#) | [Esempi semplici di programmazione](#) ]

## Esempio: utilizzare le code messaggi (parte 1 di 3)

Utilizzare quanto segue come esempio per il proprio programma. L'esempio include informazioni dettagliate sulle righe chiave nel codice. E' possibile visualizzare le informazioni dettagliate nei seguenti modi:


- Fare clic sull'immagine  per visualizzare le informazioni dettagliate in una finestra a comparsa.
- Fare clic sul collegamento testo per visualizzare le informazioni dettagliate fornite alla fine dell'esempio.

**Nota:** leggere l'[Esonero di responsabilità per gli esempi di codice](#) per informazioni legali importanti.

```
////////////////////////////////////  
//  
// 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; Nota 1   
  
import java.io.*;  
import java.util.*;  
  
import com.ibm.as400.access.*; Nota 2   
  
public class displayMessages extends Object  
{  
  
    public static void main(String[] parameters) Nota 3   
    {  
        displayMessages me = new displayMessages();  
        me.Main(parameters); Nota 4   
  
        System.exit(0); Nota 5   
    }  
  
    void displayMessage()  
    {  
    }  
  
    void Main(String[] parms)  
    {  
        try Nota 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 di accesso disponibili per questo programma. Le classi nel pacchetto di accesso 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 dispone di un metodo **main**; pertanto, può essere eseguita come applicazione. Per richiamare il programma, è necessario eseguire **java examples.displayMessages**. Tenere presente 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, al fine di 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, non verrà restituita la richiesta comandi al termine del programma. L'utente dovrà immettere `Ctrl-C` per ottenere una richiesta comandi.
6. Il codice di 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 degli errori. 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](#) | [Esempi semplici di programmazione](#) ]

Per visualizzare informazioni dettagliate in una finestra separata, il browser deve avere il supporto JavaScript abilitato. Se il browser non ha il JavaScript abilitato o non supporta JavaScript, fare clic sul collegamento testo **Nota** o andare alla fine della pagina per visualizzare le informazioni.


[ [Parte precedente](#) | [Parte successiva](#) | [Esempi semplici di programmazione](#) ]

## Esempio: utilizzare le code messaggi (parte 2 di 3)


Utilizzare quanto segue come esempio per il proprio programma. L'esempio include informazioni dettagliate sulle righe chiave nel codice. E' possibile visualizzare le informazioni dettagliate nei seguenti modi:

- Fare clic sull'immagine  per visualizzare le informazioni dettagliate in una finestra a comparsa.
- Fare clic sul collegamento testo per visualizzare le informazioni dettagliate fornite alla fine dell'esempio.

**Nota:** leggere l'[Esonero di responsabilità per gli esempi di codice](#) per informazioni legali importanti.

```
////////////////////////////////////  
//  
// 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 stabilire il server a cui 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 sia il nome del server. Se un parametro viene inviato al programma, il metodo **setSystemName** dell'oggetto AS400 viene utilizzato per impostare il nome 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](#) | [Esempi semplici di programmazione](#) ]

Per visualizzare informazioni dettagliate in una finestra separata, il browser deve avere il supporto JavaScript abilitato. Se il browser non ha il JavaScript abilitato o non supporta JavaScript, fare clic sul collegamento testo **Nota** o andare alla fine della pagina per visualizzare le informazioni.

[ [Parte precedente](#) | [Esempi semplici di programmazione](#) ]

## Esempio: utilizzare le code messaggi (parte 3 di 3)


Utilizzare quanto segue come esempio per il proprio programma. L'esempio include informazioni dettagliate sulle righe chiave nel codice. E' possibile visualizzare le informazioni dettagliate nei seguenti modi:


- Fare clic sull'immagine  per visualizzare le informazioni dettagliate in una finestra a comparsa.
- Fare clic sul collegamento testo per visualizzare le informazioni dettagliate fornite alla fine dell'esempio.


**Nota:** leggere l'[Esonero di responsabilità per gli esempi di codice](#) per informazioni legali importanti.


```
////////////////////////////////////  
//  
// 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.getMessage(); 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'obiettivo di questo programma consiste nel visualizzare i messaggi in una coda messaggi 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 una lista di messaggi dal server. Viene effettuato un collegamento al server a questo punto.
3. Rimuovere un messaggio dalla lista. Il messaggio si trova nell'oggetto QueuedMessage del programma IBM Toolbox per Java.
4. Stampare il testo del messaggio.

[ [Parte precedente](#) | [Esempi semplici di programmazione](#) ]




Per visualizzare informazioni dettagliate in una finestra separata, il browser deve avere il supporto JavaScript abilitato. Se il browser non ha il JavaScript abilitato o non supporta JavaScript, fare clic sul collegamento testo **Nota** o andare alla fine della pagina per visualizzare le informazioni.

[ [Parte successiva](#) | [Esempi semplici di programmazione](#) ]

## Esempio: utilizzare l'accesso a livello record (parte 1 di 2)

Utilizzare quanto segue come esempio per il proprio programma. L'esempio include informazioni dettagliate sulle righe chiave nel codice. E' possibile visualizzare le informazioni dettagliate nei seguenti modi:

- Fare clic sull'immagine  per visualizzare le informazioni dettagliate in una finestra a comparsa.
- Fare clic sul collegamento testo per visualizzare le informazioni dettagliate fornite alla fine dell'esempio.

**Nota:** leggere l'[Esonero di responsabilità per gli esempi di codice](#) per informazioni legali importanti.


```
////////////////////////////////////  
//  
// Esempio di accesso al livello record. Questo programma richiederà  
// all'utente  
// il nome del server ed il file da visualizzare. Il file deve esistere  
// e dovrebbe contenere dei record. Ogni record nel file verrà visualizzato  
// in System.out.  
//  
// Sintassi di chiamata: java RLSequentialAccessExample  
//  
// Questo sorgente è un esempio di "RecordLevelAccess" di 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 modulo nome percorso IFS (integrated file system) dell'oggetto che deve essere visualizzato.
3. Questa istruzione crea un oggetto che rappresenta un file sequenziale esistente sul server a cui si è connessi. Questo file sequenziale è il file che verrà visualizzato.
4. Queste righe richiamano il formato record del file.
5. Questo riga imposta il formato record per il file.
6. Questa riga apre il file selezionato per la lettura. 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 a livello record.


[ [Parte successiva](#) | [Esempi semplici di programmazione](#) ]

Per visualizzare informazioni dettagliate in una finestra separata, il browser deve avere il supporto JavaScript abilitato. Se il browser non ha il JavaScript abilitato o non supporta JavaScript, fare clic sul collegamento testo **Nota** o andare alla fine della pagina per visualizzare le informazioni.


[ [Parte precedente](#) | [Esempi semplici di programmazione](#) ]

## Esempio: utilizzare l'accesso a livello record (parte 2 di 2)

Utilizzare quanto segue come esempio per il proprio programma. L'esempio include informazioni dettagliate sulle righe chiave nel codice. E' possibile visualizzare le informazioni dettagliate nei seguenti modi:

- Fare clic sull'immagine  per visualizzare le informazioni dettagliate in una finestra a comparsa.
- Fare clic sul collegamento testo per visualizzare le informazioni dettagliate fornite alla fine dell'esempio.


**Nota:** leggere l'[Esonero di responsabilità per gli esempi di codice](#) per informazioni legali importanti.

```
////////////////////////////////////  
//  
// 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 del codice che sono 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 etichettati "Inizio nota due" e "Fine nota due" mostra come creare un formato record invece di ottenere il formato record da un file esistente. L'ultima riga in questo blocco crea il file sul server.
3. Il testo nei commenti Java etichettato "Inizio nota tre" e "Fine nota tre" mostra un modo per creare un record e successivamente scriverlo su un file.


[ [Parte precedente](#) | [Esempi semplici di programmazione](#) ]

Per visualizzare informazioni dettagliate in una finestra separata, il browser deve avere il supporto JavaScript abilitato. Se il browser non ha il JavaScript abilitato o non supporta JavaScript, fare clic sul collegamento testo **Nota** o andare alla fine della pagina per visualizzare le informazioni.

[ [Parte successiva](#) | [Esempi semplici di programmazione](#) ]

## Esempio: utilizzare le classi JDBC per creare e popolare una tabella (parte 1 di 2)

Utilizzare quanto segue come esempio per il proprio programma. L'esempio include informazioni dettagliate sulle righe chiave nel codice. E' possibile visualizzare le informazioni dettagliate nei seguenti modi:

- Fare clic sull'immagine  per visualizzare le informazioni dettagliate in una finestra a comparsa.
- Fare clic sul collegamento testo per visualizzare le informazioni dettagliate fornite alla fine dell'esempio.

**Nota:** leggere l'[Esonero di responsabilità per gli esempi di codice](#) per informazioni legali importanti.

```
////////////////////////////////////  
//  
// Esempio di 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. Una 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](#) | [Esempi semplici di programmazione](#) ]

Per visualizzare informazioni dettagliate in una finestra separata, il browser deve avere il supporto JavaScript abilitato. Se il browser non ha il JavaScript abilitato o non supporta JavaScript, fare clic sul collegamento testo **Nota** o andare alla fine della pagina per visualizzare le informazioni.

[ [Parte precedente](#) | [Esempi semplici di programmazione](#) ]

## Esempio: utilizzare le classi JDBC per creare e popolare una tabella (parte 2 di 2)

Utilizzare quanto segue come esempio per il proprio programma. L'esempio include informazioni dettagliate sulle righe chiave nel codice. E' possibile visualizzare le informazioni dettagliate nei seguenti modi:

- Fare clic sull'immagine  per visualizzare le informazioni dettagliate in una finestra a comparsa.
- Fare clic sul collegamento testo per visualizzare le informazioni dettagliate fornite alla fine dell'esempio.

**Nota:** leggere l'[Esonero di responsabilità per gli esempi di codice](#) per informazioni legali importanti.

```
////////////////////////////////////
//
// Esempio di 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 troncatura 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( " " );
            System.out.println("Usage:");
            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](#) 

```

        // Stabilire un collegamento al database. Poichè non vengono
        forniti
        // 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](#) | [Esempi semplici di programmazione](#) ]

Per visualizzare informazioni dettagliate in una finestra separata, il browser deve avere il supporto JavaScript abilitato. Se il browser non ha il JavaScript abilitato o non supporta JavaScript, fare clic sul collegamento testo **Nota** o andare alla fine della pagina per visualizzare le informazioni.

[ [Esempi semplici di programmazione](#) ]

## Esempio: visualizzare una lista di lavori server in una GUI


Utilizzare quanto segue come esempio per il proprio programma. L'esempio include informazioni dettagliate sulle righe chiave nel codice. E' possibile visualizzare le informazioni dettagliate nei seguenti modi:


- Fare clic sull'immagine  per visualizzare le informazioni dettagliate in una finestra a comparsa.
- Fare clic sul collegamento testo per visualizzare le informazioni dettagliate fornite alla fine dell'esempio.


**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 finestra di dialogo di errore.  Esso
visualizzerà all'utente qualsiasi errore.
    ErrorDialogAdapter errorHandler = new ErrorDialogAdapter (frame); Note
9 


        // Creare un pannello explorer per presentare la lista 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 ha 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 le eccezioni che devono essere rilevate dal programma.
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 una lista di lavori server che possono essere visualizzati in un componente (GUI) vaccess. Notare che l'oggetto AS400 viene utilizzato per specificare il server su cui risiede la lista.
8. Questa riga crea una frame o una finestra di livello superiore che verrà utilizzata per visualizzare la lista 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 inzializza 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 (9) come un listener sul componente GUI di 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 ad 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 della lista di lavori al centro della frame di controllo.
15. Questa riga richiama il metodo show per rendere la finestra visibile all'utente.
16. Le eccezioni IBM Toolbox per Java vengono convertite in modo tale che il testo appaia nella lingua della stazione di lavoro. Ad esempio, questo programma visualizza il testo dell'eccezione come errore di elaborazione.
17. 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 comandi DOS di Windows 95 senza questa riga, non verrà restituita la richiesta dei comandi alla chiusura del programma.

[ [Esempi semplici di programmazione](#) ]



# Esempi: suggerimenti per la programmazione

Questa sezione elenca gli esempi di codice forniti per tutta la documentazione relativa all'argomento sulla gestione dei collegamenti.

## Gestire i collegamenti

- [Esempio: effettuare un collegamento al server iSeries con un oggetto CommandCall](#)
- [Esempio: effettuare due collegamenti al server iSeries con un oggetto CommandCall](#)
- [Esempio: creare oggetti CommandCall e IFSFileInputStream con un oggetto AS400](#)
- [Esempio: utilizzare AS400ConnectionPool per precollegarsi al server iSeries](#)
- [Esempio: utilizzare AS400ConnectionPool per precollegarsi ad un servizio specifico sul server iSeries, quindi riutilizzare il collegamento.](#)

## Avviare e chiudere 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: utilizzare le eccezioni](#)

## Eventi di errore

- [Esempio: gestire gli eventi di errore](#)
- [Esempio: definire un listener di errori](#)
- [Esempio: utilizzare un handler personalizzato per gestire eventi di errori](#)

## Traccia

- [Esempio: utilizzare la traccia](#)
- [Esempio: utilizzare setTraceOn\(\)](#)
- [Esempio: utilizzare la traccia componente](#)

## Ottimizzazione

- [Esempio: creare due oggetti AS400](#)
- [Esempio: utilizzare un oggetto AS400 per rappresentare un secondo server](#)

# Installare e aggiornare

- [Esempio: utilizzare la classe AS400Toolbox Installer](#)

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: ToolboxME per iSeries

Questa sezione elenca gli esempi di codice che sono forniti in tutto ToolboxME per la documentazione iSeries.

- [Esempio: creare un esempio ToolboxME per iSeries - JdbcDemo.java](#)
- [Esempio: utilizzare ToolboxME per iSeries, MIDP e JDBC](#)
- [Esempio: utilizzare ToolboxME per iSeries, MIDP e Toolbox per Java](#)

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. Le garanzie assicurate di non violazione, commerciabilità ed idoneità ad un particolare scopo sono smentite espressamente. «



# Esempio: utilizzare 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 di ToolboxME per iSeries. Questo programma è un MIDlet di esempio  
// che mostra come  
// si potrebbe 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
        // nella lista.
        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;
} // 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're 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() + ";" +
jdbcmText.getString());

        stmt = conn.createStatement();

        // Poichè 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're interested in.
*/
public void submitBid()
{
    java.sql.Connection    conn = null;
    Statement              stmt = null;

    try
    {
        conn = DriverManager.getConnection(urlText.getString() + ";" +
jdbcmeText.getString());

        stmt = conn.createStatement();

        // Poichè 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 la lista attiva 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 don't need right now.
 */
public void pauseApp()
{
    display.setCurrent(null);
}

/**
 * Destroy must cleanup everything.
 */
public void destroyApp(boolean unconditional)
{
}
}◀◀
```



# Esempio: utilizzare ToolboxME per iSeries, MIDP e 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 di 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("Toolbox 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);
private TextField signonPwdText_ = new TextField("Password", "JTEAM1",
10, TextField.PASSWORD); // TBD temporaneo
private TextField signonServerText_ = new TextField("MEServer",
"localhost", 10, TextField.ANY);
private StringItem signonStatusText_ = new StringItem("Status",
NOT_SIGNED_ON);

// TextField per il modulo CommandCall.
private TextField cmdText_ = new TextField("Command", "CRTLIB FRED",
256, TextField.ANY); // TBD: max size; TBD: TextBox???
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);

/**
 * Creates a new ToolboxMidpDemo.
 */
public ToolboxMidpDemo()
{
    display_ = Display.getDisplay(this);
    // Nota: Il demo basato su KVM ha utilizzato TabbedPane per il
pannello principale. MIDP non dispone di una classe simile, quindi si
utilizza una List invece.
}

/**
 * Show the main screen.
 * Implements abstract method of class 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);
    }
}
else // 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 nella Toolbox programmer's
guide.
        String pcmlName = "qsyrusri.pcml"; // Il file PCML che
si desidera utilizzare.
        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 le
azioni di Data Queue, visualizzare un 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
wireless 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_) // "Info su".
    {
        // Non si sarebbe dovuti mai arrivare a questo punto, poiché
        l'unico pulsante è "Indietro".
    } // Nessuno dei precedenti.
    else
        feedback("Internal error: Form is not recognized.",
AlertType.ERROR);
    } // instanceof Form
    else
        feedback("Internal error: Displayable object not recognized.",
AlertType.ERROR);
    }
}

```

```

/**
 * Displays the "About" form.
 **/
private void showAboutForm()
{
    // Se il modulo info su è nullo, crearlo e accoderlo.
    if (aboutForm_ == null)
    {
        aboutForm_ = new Form(ABOUT);
        aboutForm_.append(new StringItem(null, "This is a MIDP example
application that uses the Toolbox Micro Edition (ToolboxME)."));

        aboutForm_.addCommand(actionBack_);
        aboutForm_.setCommandListener(this);
    }

    display_.setCurrent(aboutForm_);
}

```

```

/**
 * Displays the "SignOn" form.
 **/
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_);
}

```



```

/**
 * Displays the "CommandCall" form.
 */
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_);
}

/**
 * Displays the "ProgramCall" form.
 */
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_);
}

/**
 * Displays the "DataQueue" form.
 */
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());
}

/**
 * This method is used to create a dialog and display feedback
information using an Alert to the user.
 */
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.
 * Implements abstract method of class 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 di utilità

Questa sezione elenca gli esempi di codice che sono forniti in tutta la documentazione relativa alle classi programma di utilità.

## Programma di installazione di IBM Toolbox

- [Esempio: utilizzare la classe AS400ToolboxInstaller](#)
- [Esempio: installare IBM Toolbox per Java utilizzando AS400ToolboxInstaller](#)
- [Esempio: installare il pacchetto ACCESS dalla riga comandi](#)
- [Esempio: utilizzare la classe Graphical Toolbox dalla riga comandi](#)

## JarMaker

- [Esempio: estrarre AS400.class e tutte le relative classi dipendenti da jt400.jar](#)
- [Esempio: suddividere jt400.jar in una serie di file 300K](#)
- [Esempio: eliminare i file inutilizzati da un file JAR](#)
- [Esempio: creare un file JAR più piccolo di 400K byte omettendo le tabelle di conversione con il parametro `-ccsid`](#)

## »CommandPrompter

- [Esempio: utilizzare CommandPrompter per richiamare 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**

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: utilizzare 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 trovato esso controlla 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 in 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);

                // Controllare 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. Verificare se gli aggiornamenti
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: utilizzare CommandPrompter

**Nota:** leggere l'[Esonero di responsabilità per gli esempi di codice](#) per informazioni legali importanti.

```
////////////////////////////////////  
//  
// 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.

## AS400Panels

- [Esempio: creare un AS400DetailsPane per presentare la lista degli utenti definiti in systemAS400DetailsPane](#)
- [Esempio: caricare il contenuto di un pannello di dettagli prima di aggiungerlo alla frame](#)
- [Esempio: utilizzare AS400ListPane per presentare una lista di utenti](#)
- [Esempio: utilizzare AS400DetailsPane per visualizzare i messaggi restituiti da una chiamata al comando](#)
- [Esempio: utilizzare AS400TreePane per visualizzare una vista ad albero di un indirizzario](#)
- [Esempio: utilizzare AS400ExplorerPane per presentare varie risorse di stampa](#)

## Command call

- [Esempio: creare CommandCallButton](#)
- [Esempio: aggiungere ActionListener per elaborare tutti i messaggi iSeries che vengono creati da un comando](#)
- [Esempio: utilizzare CommandCallMenuItem](#)

## Code di dati

- [Esempio: creare DataQueueDocument](#)
- [Esempio: utilizzare DataQueueDocument](#)

## Eventi di errore

- [Esempio: gestire gli eventi errore](#)
- [Esempio: definire un listener di errori](#)
- [Esempio: utilizzare un handler personalizzato per gestire eventi errore](#)

## JDBC

- [Esempio: utilizzare l'unità di controllo JDBC per creare e popolare una tabella](#)
- [Esempio: utilizzare l'unità di controllo JDBC per interrogare una tabella ed emettere il suo contenuto](#)
- [Esempio: creare AS400JDBCDataSourcePane](#)

## Lavori

- [Esempio: creare un VJobList e presentare la lista in un AS400ExplorerPane](#)
- [Esempio: presentare una lista di lavori in un pannello di ricerca](#)

## Chiamata al programma

- [Esempio: creare un ProgramCallMenuItem](#)
- [Esempio: elaborare tutti i messaggi iSeries generati dal programma](#)
- [Esempio: aggiungere due parametri](#)
- [Esempio: utilizzare un ProgramCallButton in un'applicazione](#)

## Accesso al livello record

- [Esempio: creare un oggetto RecordListTablePane per visualizzare tutti record inferiori o uguali ad una chiave](#)

## SpooledFileViewer

- [Esempio: creare un programma di visualizzazione del file di spool per visualizzare un file di spool creato precedentemente su iSeries](#)

## Valori di sistema

- [Esempio: creare una GUI del valore di sistema utilizzando il pannello AS400Explorer](#)

## Utenti e gruppi

- [Esempio: creare un VUserList con AS400DetailsPane](#)
- [Esempio: utilizzare un AS400ListPane per creare una lista 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**

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: utilizzare VUserList

**Nota:** leggere l'[Esonero di responsabilità per gli esempi di codice](#) per informazioni legali importanti.

```
////////////////////////////////////  
//  
// Esempio di VUserList. Questo programma presenta una lista di utenti su  
// un sistema in un pannello lista 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 una VUserList. Questa rappresenta una lista di utenti  
            // visualizzati nel pannello lista.  
            VUserList userList = new VUserList (system);  
  
            // Creare una frame.  
            JFrame f = new JFrame ("VUserList example");  
  
            // Creare un adattatore di finestra di dialogo di errore. E esso  
visualizza  
            // qualsiasi errore all'utente.  
            ErrorDialogAdapter errorHandler = new ErrorDialogAdapter (f);  
  
            // Creare un pannello lista per visualizzare la lista 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 lista.
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: utilizzare VMessageList

**Nota:** leggere l'[Esonero di responsabilità per gli esempi di codice](#) per informazioni legali importanti.

```
////////////////////////////////////  
//  
// 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 di finestra di dialogo di errore.  
Questo visualizzerà  
            // qualsiasi errore all'utente.  
            ErrorDialogAdapter errorHandler = new ErrorDialogAdapter (f);
```

```

        // Creare un pannello di dettagli per visualizzare la lista 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: utilizzare VIFSDirectory

**Nota:** leggere l'[Esonero di responsabilità per gli esempi di codice](#) per informazioni legali importanti.

```
////////////////////////////////////
//
// 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 di finestra di dialogo di 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: utilizzare VPrinters

**Nota:** leggere l'[Esonero di responsabilità per gli esempi di codice](#) per informazioni legali importanti.

```
////////////////////////////////////
//
// Esempio VPrinters.  Questo programma presenta varie risorse
// stampa di rete con un pannello explorer.
//
// Sintassi del comando:
//   VPrintersExample system
//
// Questo sorgente è un esempio di "VPrinters" 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 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 la lista
            // di stampanti collegate al sistema.
            VPrinters printers = new VPrinters (system);

            // Creare una frame.
            JFrame f = new JFrame ("VPrinters example");

            // Creare un adattatore di finestra di dialogo di 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: utilizzare CommandCallMenuItem

**Nota:** leggere l'[Esonero di responsabilità per gli esempi di codice](#) per informazioni legali importanti.

```
////////////////////////////////////  
//  
// Esempio voce di menu chiamata al comando. Questo programma dimostra come  
utilizzare  
// una voce menu che chiama un comando server. Esso visualizzerà  
// qualsiasi messaggio restituito 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 stato 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.  
Questo visualizzerà  
            // 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 la lista dei messaggi dall'origine evento.
        CommandCallMenuItem item = (CommandCallMenuItem)
event.getSource ();
        AS400Message[] messageList = item.getMessageList ();

        // Utilizzare un AS400DetailsPane per visualizzare i
messaggi.
        VMessaggeList vmessaggeList = new VMessaggeList
(messageList);
        AS400DetailsPane messageDetails = new AS400DetailsPane
(vmessaggeList);
        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: utilizzare DataQueueDocument

**Nota:** leggere l'[Esonero di responsabilità per gli esempi di codice](#) per informazioni legali importanti.

```
////////////////////////////////////
//
// Esempio di DataQueueDocument. Questo programma dimostra 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.
            Questo visualizzerà
            // qualsiasi errore all'utente.
            ErrorDialogAdapter errorHandler = new ErrorDialogAdapter (f);

            // Creare un adattatore del cursore di lavoro. Questo regolerà
            // 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 stato 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 le 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
// hanno luogo letture e scritture. Si consentirà
// all'utilizzo di 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);
        }
    }
}
```

# 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().

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**.

## » Esempio: utilizzare AS400JDBCDataSourcePane

```
// 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 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
        applicate con
        // all'origine dati. Se tutte le modifiche vengono
        // 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: utilizzare VJobList per presentare una lista di lavori

**Nota:** leggere l'[Esonero di responsabilità per gli esempi di codice](#) per informazioni legali importanti.

```
////////////////////////////////////  
//  
// Esempio di lista lavori. Questo programma presenta una lista 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 la lista  
            // 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 di finestra di dialogo di errore.  
            // Questo visualizzerà  
            // qualsiasi errore all'utente.  
            ErrorDialogAdapter errorHandler = new ErrorDialogAdapter (f);  
  
            // Creare un pannello explorer per presentare la lista 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: utilizzare un pulsante per chiamare un programma sul server

**Nota:** leggere l'[Esonero di responsabilità per gli esempi di codice](#) per informazioni legali importanti.

```
////////////////////////////////////  
//  
// Esempio pulsante chiamata al programma. Questo programma dimostra come  
// utilizzare un pulsante che chiami un programma sul server. Questo  
// scambierà i 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-static di main(). Se non viene effettuata questa  
    chiamata  
    // le variabili di classe (parm1, parm2, ...) devono essere dichiarate  
    // static. 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 di finestra di dialogo di 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 impostato 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 quello relativo alle informazioni
sull'errore. Esso

```

```

// è 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.addActionListener (new ActionListener
()
{
    public void actionPerformed (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 = parmlConverter.toInt (parmlBytes, 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 = parmlConverter.toInt (parmlBytes, 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);
        }
}
}

```

# Suggerimenti per la programmazione

Questa sezione presenta una varietà di suggerimenti per l'utilizzo di IBM Toolbox per Java:

## [Chiudere il programma Java](#)

Acquisire informazioni su come chiudere il programma Java in modo appropriato.

## [Utilizzare i 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.

## [Gestire i collegamenti](#)

Scoprire come utilizzare la classe AS400 per avviare e chiudere i collegamenti socket e come renderli compatibili con la specifica Enterprise JavaBean.

## [Utilizzare la 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.

## [»Effettuare il collegamento ad uno IASP \(independent auxiliary storage pool\)](#)

Consultare le informazioni sul collegamento ad uno IASP. Uno IASP è una raccolta di unità disco che è possibile collegare e scollegare indipendentemente dal resto della memoria su un sistema.◀

## [Gestire gli errori durante l'utilizzo delle classi access di Toolbox per Java](#)

Acquisire informazioni su come utilizzare le classi di eccezioni Toolbox per Java per gestire gli errori quando si utilizzano le classi access Toolbox per Java nel proprio programma.

## [Gestire gli errori durante l'utilizzo delle classi vaccess di Toolbox per Java](#)

Acquisire informazioni su come utilizzare le classi eventi errore Toolbox per Java per gestire gli errori quando si utilizzano le classi vaccess Toolbox per Java nel proprio programma.

## [Utilizzare la 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.

## [Ottimizzare i propri programmi](#)

Acquisire informazioni su come ottimizzare i propri programmi per delle prestazioni migliori.

## [Ottenere delle prestazioni migliori utilizzando JVM di OS/400](#)

Consultare le informazioni sulle prestazioni potenziate, conseguenza dell'utilizzo di JVM di OS/400.

## [Gestire le classi di Toolbox per Java sul proprio client](#)

Acquisire conoscenze su come utilizzare la classe AS400ToolboxInstaller per gestire le classi IBM Toolbox per Java sul proprio client.

## [Migliorare le prestazione del file JAR](#)

Scoprire come utilizzare la classe JarMaker di Toolbox per Java per creare file JAR di IBM Toolbox per Java di dimensioni più ridotte e a caricamento più veloce.

## [Utilizzare il national language support di Java](#)

Consultare le informazioni sull'utilizzo di IBM Toolbox per Java e dell'NLS di Java.

## [Richiedere servizio e supporto](#)

Utilizzare queste risorse per individuare i servizi di supporto per Toolbox per Java.

# Chiudere il 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 perché potrebbero causare la chiusura dell'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 sì 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 o AS/400e.

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 <b>necessaria</b> di un nome percorso IFS. Il nome della libreria deve essere composto da un massimo di 10 caratteri seguito da <b>.lib</b> .
oggetto	Il nome dell'oggetto rappresentato dal nome percorso IFS. L'oggetto è una parte <b>necessaria</b> di un nome percorso IFS. Il nome dell'oggetto deve essere composto da un massimo di 10 caratteri e seguito da <b>.type</b> , dove <b>type</b> 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' <b>oggetto</b> . (Si rimanda alla voce <b>oggetto</b> .) 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 <b>facoltativa</b> di un nome percorso IFS. Può essere specificato solo quando il <b>tipo oggetto</b> è <b>FILE</b> . Il nome del membro deve contenere un massimo di 10 caratteri e deve essere seguito da <b>.mbr</b> .

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 una lista 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 una lista 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 classi Toolbox per Java, il formato per questi valori speciali inizia e termina con il simbolo di percentuale (%**ALL**%).

**Nota:** nell'IFS, un asterisco è 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 Toolbox per Java.

Componente nome percorso	Formato tradizionale	Formato 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](#).

# Gestire i 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 ad un sistema 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 abilita i 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:

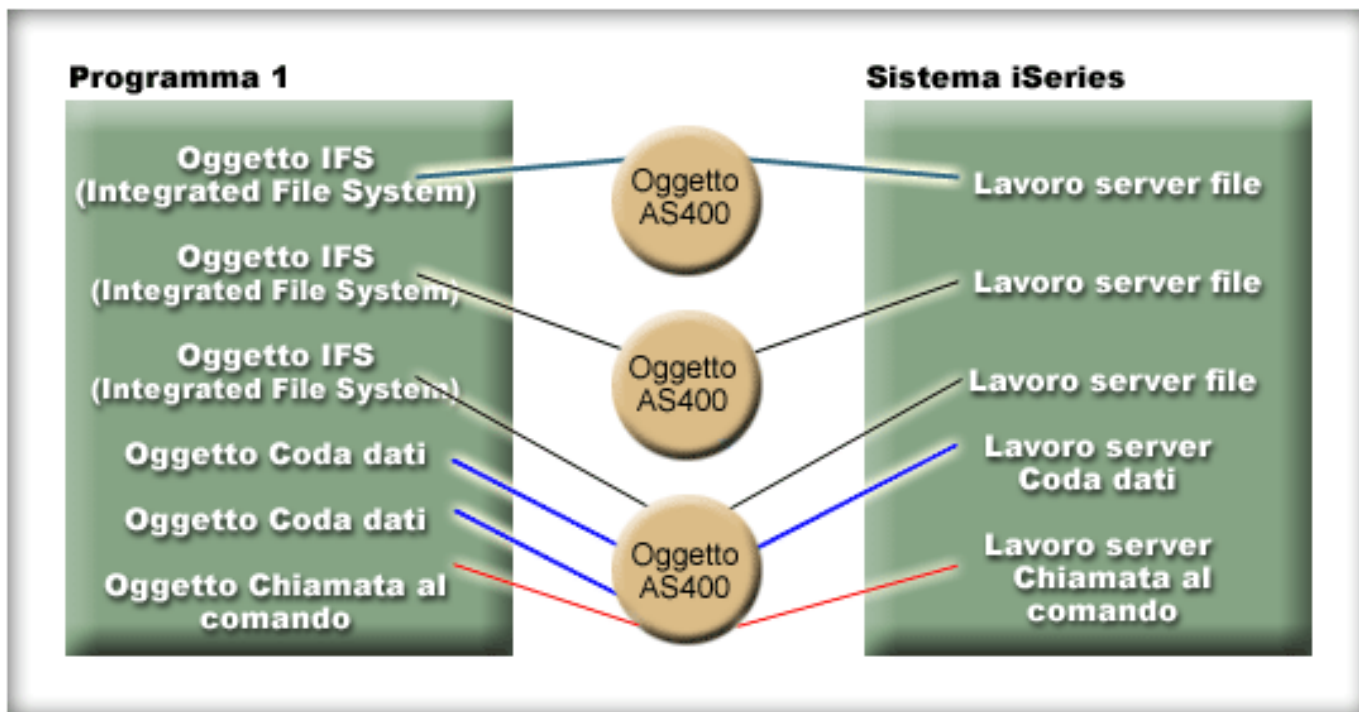
- JDBC
- Chiamata al programma e chiamata al comando
- IFS (Integrated file system)
- Stampa
- Coda dati
- Accesso al livello record

## Note:

- 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.

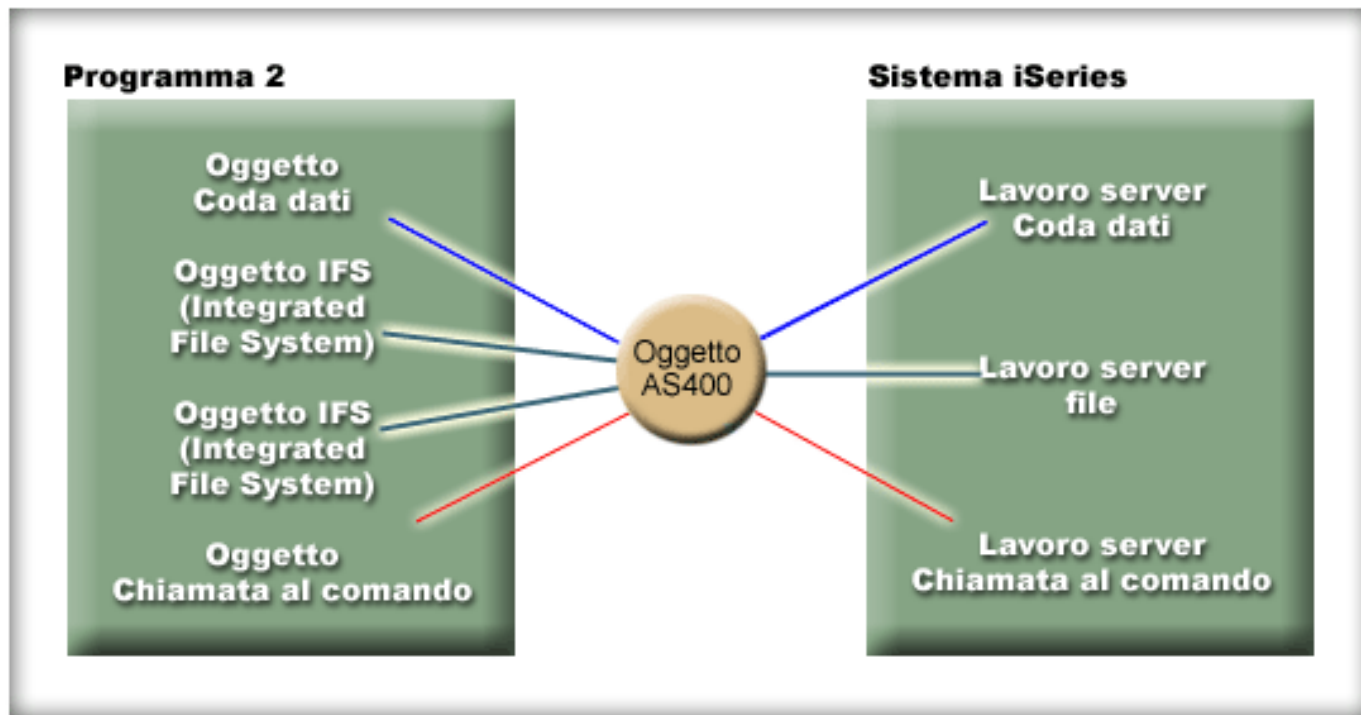
Il programma Java può controllare il numero di collegamenti a iSeries. Per ottimizzare le prestazioni delle comunicazioni, un programma Java può creare più oggetti AS400 per lo stesso sistema come mostrato in Figura 1. Ciò crea più collegamenti socket con iSeries.

**Figura 1: il programma Java crea più oggetti AS400 e collegamenti socket per lo stesso sistema iSeries**



Per conservare le risorse del sistema iSeries, creare solo un oggetto AS400 come mostrato in Figura 2. Questo approccio riduce il numero di collegamenti, il che riduce a sua volta la quantità di risorse utilizzate nel sistema.

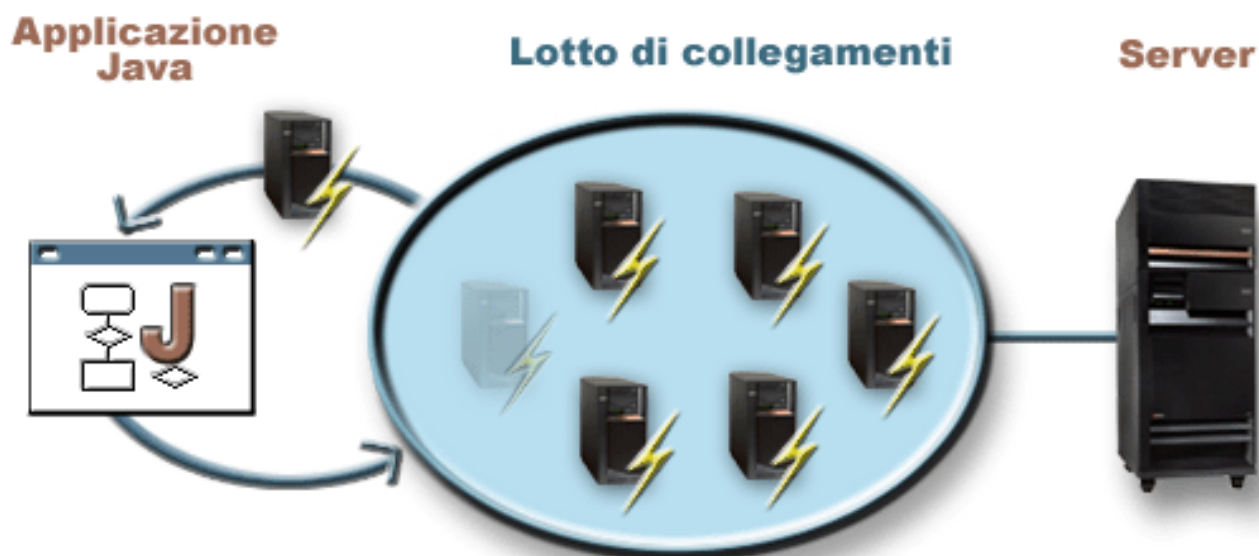
**Figura 2: il programma Java crea un singolo oggetto AS400 ed un collegamento socket per lo stesso sistema iSeries**



**Nota:** anche se la creazione di più collegamenti aumenta la quantità di risorsa utilizzata 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 in Figura 3. Questo approccio riduce la quantità di tempo necessaria per collegarsi ad iSeries riutilizzando un collegamento precedentemente stabilito per l'utente.

**Figura 3: il programma Java richiama un collegamento da un AS400ConnectionPool in 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 sistema iSeries. Poiché gli oggetti CommandCall utilizzano lo stesso oggetto AS400, viene creato un solo collegamento al sistema.

```
// 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 quando // viene eseguito il primo 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 comandi allo stesso sistema iSeries. Poiché gli oggetti CommandCall utilizzano differenti oggetti di AS400, vengono creati due collegamenti al sistema.

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

// Creare due oggetti di chiamata al comando. Essi
usano // diversi oggetti AS400.
CommandCall cmd1 = new CommandCall(sys1,"myCommand1");
CommandCall cmd2 = new CommandCall(sys2,"myCommand2");

// Eseguire i comandi. Viene stabilito un
collegamento quando // viene eseguito il primo comando. Poiché il secondo
oggetto di // comando usa 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 sistema 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. La sua creazione farà in modo
che // l'oggetto AS400 si colleghi al servizio file.
IFSFileInputStream file = new IFSFileInputStream(newConn1,"/myfile");
```



```

collegamento al          // Eseguire il comando. Viene stabilito un
                           // 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");
collegamento al          // Eseguire il comando. Viene stabilito un
                           // 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 di servizio
    // definite nella classe AS400 (FILE, PRINT, COMMAND, DATAQUEUE
ecc.))
    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");
collegamento            // Eseguire il comando. E' già stato stabilito un
                           // al servizio comando.
    cmd.run();
                           // Restituire il collegamento al lotto.
    testPool1.returnConnectionToPool(newConn1);
                           // Richiamare un altro collegamento al servizio
comando. In questo caso, restituirà lo
                           // stesso collegamento come in precedenza, indicando
che non sarà necessario tempo di collegamento aggiuntivo ora o quando
                           // viene utilizzato il servizio comando.
    AS400 newConn2 = testPool1.getConnection("myAS400", "myUserID",
"myPassword", AS400.COMMAND);

```

## Avviare e chiudere 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:

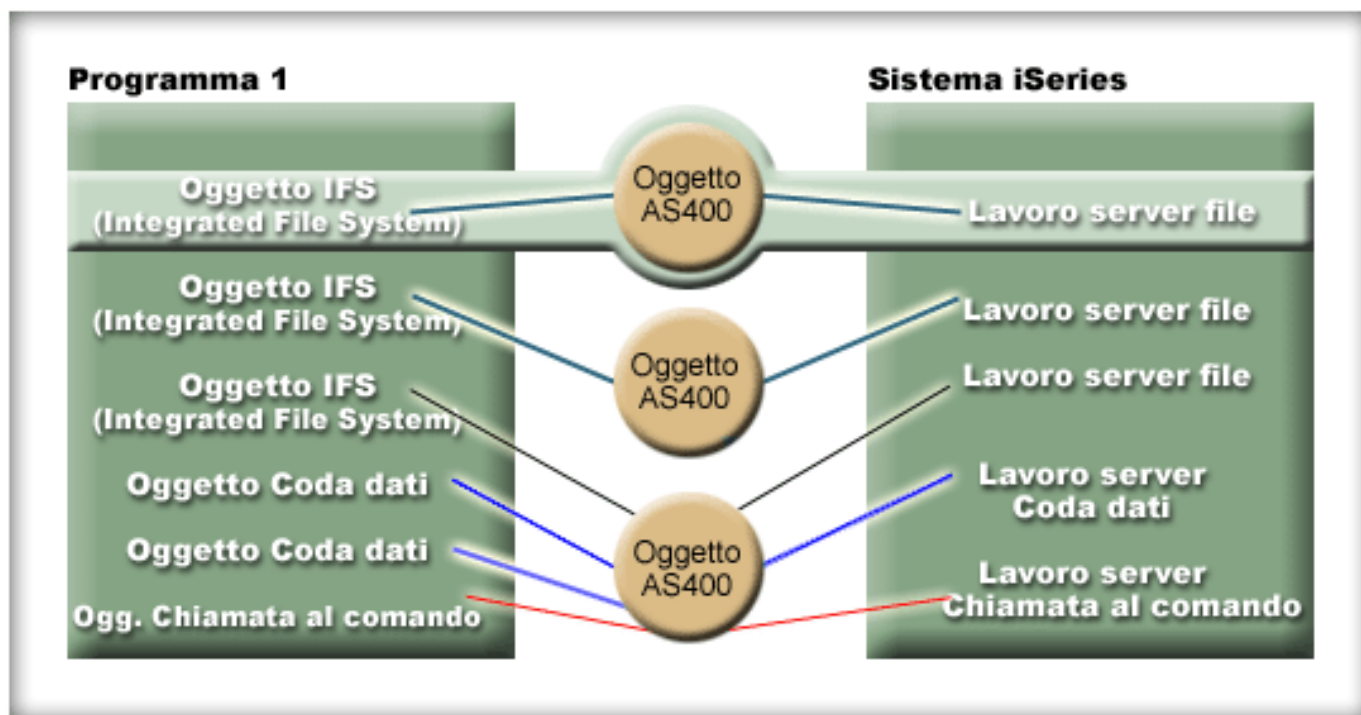
```
// Creare un oggetto AS400.
AS400 system1 = new AS400("mySystem.myCompany.com");

// Collegarsi al servizio comando. Farlo ora e non
// quando i dati vengono inviati per la prima volta
// al servizio comando. E' facoltativo in quanto
// 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");

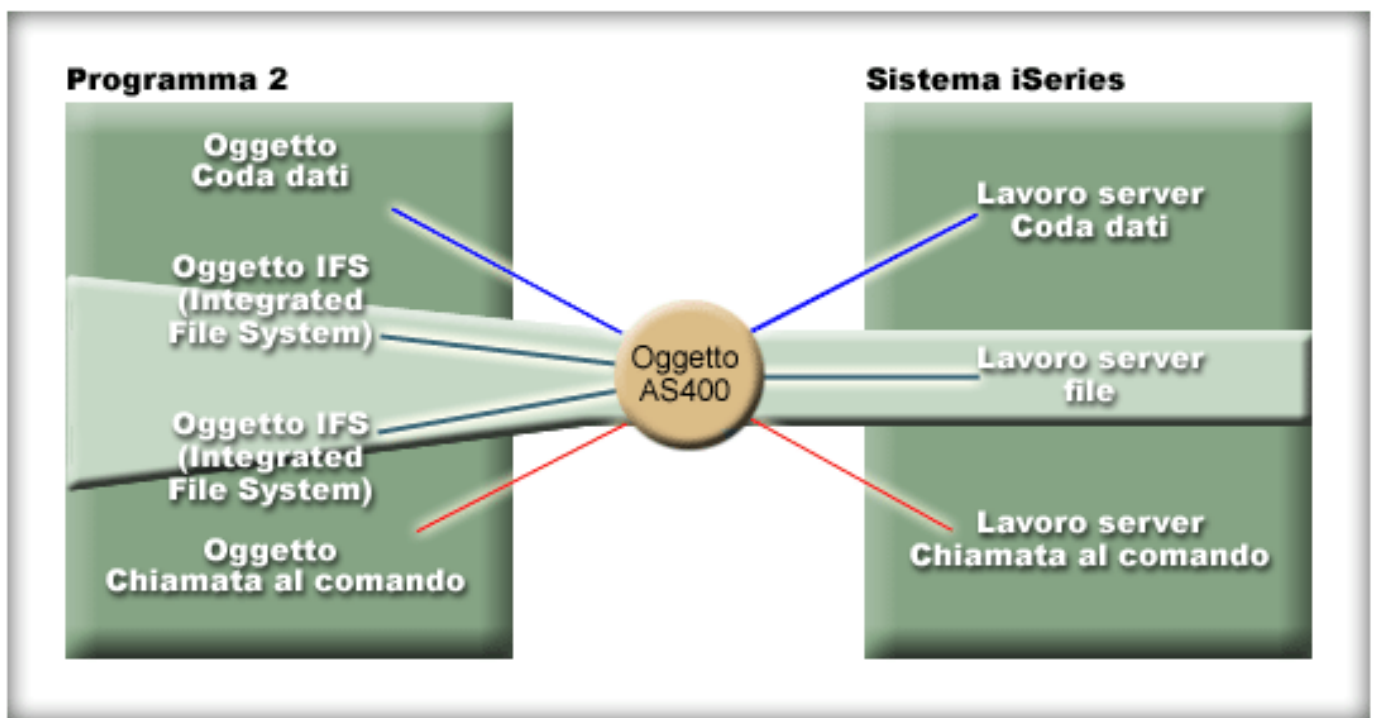
// ... usare la chiamata al comando per inviare
diversi comandi
// al server. Poiché connectService() non è stato
richiamato,
// l'oggetto AS400 si collegherà automaticamente
// quando viene eseguito il primo comando.

// 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 creato 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. Questa
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
        // usi 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);

```

# 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 si eseguono su ogni piattaforma che supporti JDK (Java Development Kit) 1.1.x e le specifiche J2SDK (Java 2 Software Development Kit).

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 [Eseguire le 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 del server iSeries per differenti piattaforme Java, consultare [Support for multiple JDKs](#) in [IBM Developer Kit per Java](#).

# Effettuare un 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 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.

# Eseguire le 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):

## 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.

## Chiamata al comando

Due modi comuni di richiamare un comando sono i seguenti:

- La classe CommandCall di IBM Toolbox per Java
- `java.lang.runtime.exec()`

La classe [CommandCall](#) crea una lista di messaggi disponibili per un programma Java al completamento del comando. Questa lista di messaggi non è disponibile tramite `java.lang.runtime.exec()`.

`java.lang.runtime.exec()` è compatibile con molte piattaforme, così, se il programma deve accedere ai file su diversi tipi di server, `java.lang.runtime.exec()` rappresenta 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 delle 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.

# Impostare il nome di sistema, l'ID utente e la 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 Rilascio 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.

I seguenti esempi mostrano come utilizzare l'oggetto AS400 con JVM OS/400.

**Esempio 1:** quando un programma Java viene eseguito sulla JVM OS/400, non è necessario che il programma fornisca un nome di sistema, un ID utente o una parola d'ordine.

**E' necessario che la parola d'ordine sia fornita quando si utilizza l'accesso al 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 nella
// JVM OS/400, la funzionalità dei due oggetti è la stessa.
// Essi si collegheranno al server corrente utilizzando l'ID utente e
```

```
// parola d'ordine del lavoro che ha avviato il programma Java.  
AS400 sys = new AS400()  
AS400 sys2 = new AS400("localhost")
```

**Esempio 2:** il programma Java può impostare un>ID utente e una parola d'ordine anche quando il programma viene eseguito 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. Collegarsi al server corrente ma non  
utilizzare  
// l>ID utente e la parola d'ordine del lavoro che ha avviato il  
// programma. Vengono utilizzati i valori forniti.  
AS400 sys = new AS400("localhost", "USR2", "PSWRD2")
```

**Esempio 3:** un programma Java in esecuzione su un server può collegarsi ad altri sistemi iSeries ed utilizzarne le risorse.

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 sul 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")
```



## 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 sistemi e i dati del sistema stesso. Quindi, mentre il sistema è attivo, è possibile collegare l'IASP disattivo o effettuare uno switch tra i sistemi.

Per ulteriori informazioni, consultare [IASP](#) e [ASP utente](#).

E' possibile utilizzare "[nome database](#)" proprietà JDBC o il metodo [setDatabaseName\(\)](#) dalla classe AS400JDBCDataSource per specificare l'IASP 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 iSeries. OS/400 ha incorporato le classi supplementari a partire dalla Versione 4 Rilascio 3.

## Abilitare le 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 supplementari fornite con OS/400, i programmi Java hanno opzioni aggiuntive per fornire informazioni sul nome di sistema, sull'ID utente e sulla parola d'ordine ad IBM Toolbox per Java.

Durante l'accesso ad una risorsa iSeries, le classi IBM Toolbox per Java devono avere un nome di sistema, un ID utente ed 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 supplementari, l'ID utente e la parola d'ordine del lavoro corrente possono essere utilizzati anche quando un programma Java in esecuzione su un iSeries accede alle risorse su un altro 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");
```

Nel seguente esempio vengono creati due oggetti AS400. I due oggetti hanno lo stesso funzionamento: entrambi eseguono un comando sull'iSeries 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 viene usato 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
// parola d'ordine 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. Questo oggetto invia
richieste
// all'iSeries locale utilizzando l'ID utente e la
parola
// d'ordine del lavoro. Questo 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();
```

Nel seguente esempio viene creato un oggetto AS400 che rappresenta un secondo sistema iSeries. Poiché si utilizza "\*current", l'ID utente e la parola d'ordine del lavoro ricavati dall'iSeries che esegue il programma Java vengono utilizzate sul secondo sistema iSeries (di destinazione).

```

// Creare un oggetto AS400. Questo oggetto invia
richieste
// ad un secondo iSeries utilizzando l'ID utente e la
// parola d'ordine del lavoro sull'iSeries corrente.
AS400 sys = new AS400("mySystem.myCompany.com", "*current",
```

```
"*current");
```

```
un comando // Creare un oggetto chiamata al comando per eseguire  
// sull'iSeries corrente.  
CommandCall cmd = new CommandCall(sys,"myCommand1");  
  
cmd.run(); // Eseguire il comando.
```

# 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 iSeries

Le prestazioni delle seguenti classi di IBM Toolbox per Java risultano potenziate poiché queste classi richiamano direttamente le API iSeries invece di richiamare un programma del 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 la lista di porte per il server.

Per un miglioramento nella definizione porta, è stato aggiunto qualche metodo alla [classe AS400](#):



- [getServicePort](#)
- [setServicePort](#)
- [setServicePortsToDefault](#)

## Modifiche MRI

I file MRI vengono ora forniti nel programma 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() è ora più veloce perché i file MRI sono memorizzati nella prima posizione che il computer 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 gli oggetti stringa Java e i pacchetti codice iSeries.

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 la propria applicazione Java è in esecuzione sulla JVM di iSeries, è possibile **migliorare in modo significativo le prestazioni** se si crea un programma Java da un file zip o da un file 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 di 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 di 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 di risoluzione problema Toolbox per Java](#)

Utilizzare queste informazioni per risolvere problemi quando si utilizza Toolbox per Java.

## [Forum JTOpen/Toolbox per Java](#)

Unirsi alla comunità dei programmatori Java che utilizzano Toolbox per Java. Questo forum è un modo valido per ottenere assistenza e consiglio da altri programmatori Java e a volte anche dagli stessi costruttori di Toolbox per Java

## [Supporto server](#)

Utilizzare il sito web del supporto server IBM per trovare gli strumenti e le risorse che aiutano a snellire le pianificazioni tecniche e il supporto per il server iSeries.

## [Supporto software](#)


Utilizzare il sito web servizi di supporto software IBM per trovare la vasta schiera di servizi di supporto software offerta 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.

# Informazioni correlate per IBM Toolbox per Java

La lista 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 Toolbox per Java e con gli sviluppatori stessi di Toolbox per Java.



## » Risorse Micro Edition di IBM Toolbox per Java 2



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.
- [Micro Edizione, piattaforma Sun Java 2](#) : 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 una vasta gamma di informazioni tecniche per gli sviluppatori dell'applicazione 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.
- [Java for iSeries, PartnerWorld for Developers](#) : fornisce informazioni su Java e sull'utilizzo possibile ed effettivo 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 una lista dei tecnici della manutenzione disponibili.
- [iSeries 400 Directory Services \(LDAP\)](#) : fornisce informazioni sull'LDAP (Lightweight Directory Access Protocol) su OS/400.

## » JSSE (Java Secure Socket Extension)

- [JSSE \(Java Secure Socket Extension\)](#) : offre una breve panoramica su JSSE e sui collegamenti per avere maggiori 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, PartnerWorld for Developers](#) : 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.

- [XML Support on iSeries, PartnerWorld for Developers](#) : fornisce informazioni sull'XML e sull'utilizzo possibile ed effettivo che i partner aziendali IBM ne fanno.
- [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.
- [XMLephanT](#) : offre risorse utili alla conoscenza di XML compresi collegamenti a molti altri siti XML.
- [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.

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.