IBM® WebSphere® Host Publisher Programmer's Guide and Reference

Version 4.0

IBM® WebSphere® Host Publisher Programmer's Guide and Reference

Version 4.0

Note

Before using this information and the product it supports, be sure to read the general information under "Appendix A. Notices" on page 83.

Third Edition (April 2002)

© Copyright International Business Machines Corporation 2000, 2002. All rights reserved.
US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

© Copyright IBM Corp. 2000, 2002

About this information

This information is designed to help you, the Host Publisher programmer, understand how to use the functions provided by Host Publisher. This book also includes information about the file formats and macro script syntax created by Host Publisher components, which enables you to edit the files manually.

Additional information resources are available for learning to use Host Publisher features. These resources include the product README, online help, and product Web pages. (See "Where can I find information online?").

This book is available as an HTML file on the installation CD, as a PDF file on the CD, and as an HTML file on the product Web site. Visit the Web site for the most updated version of this document.

Where can I find information online?

You can find the following forms of documentation for Host Publisher online.

Manuals

To access online documentation in either Host Publisher Studio or Host Publisher Server, you can use a Web browser to open HTML files and book (PDF) files on your local system.

In Host Publisher Studio

Listed below are the locations for the files for each document, where *install_dir* is the directory in which Host Publisher is installed.

IBM WebSphere Host Publisher Administrator's and User's Guide
install_dir\Common\doc\guide\guide.htm

To open the book (PDF) version, use the file name guide.pdf instead of guide.htm.

IBM WebSphere Host Publisher Planning and Installation Guide
install_dir\Common\doc\install\instgd.htm

To open the book (PDF) version, use the file name instgd.pdf instead of instgd.htm.

IBM WebSphere Host Publisher Programmer's Guide and Reference
install_dir\Common\doc\proggd\proggd.htm

To open the book (PDF) version, use the file name progguid.pdf instead of proggd.htm.

IBM WebSphere Host Publisher Messages Reference
install_dir\Common\doc\msgref\msgref.htm

To open the book (PDF) version, use the file name msgref.pdf instead of msgref.htm.

IBM Host Publisher *Readme*
install_dir\Common\doc\readme.htm

© Copyright IBM Corp. 2000, 2002

In Host Publisher Server

In Host Publisher Server, the documentation is available in any language you choose, when WebSphere is running on the Server. Open a browser to http://ServerName/HPDoc/HPDocServlet to choose the desired language and view the documentation in that language.

Listed below are the file names for each document.

IBM WebSphere Host Publisher Administrator's and User's Guide
guide.htm and guide.pdf in \guide subdirectory

IBM WebSphere Host Publisher Planning and Installation Guide
instgd.htm and instgd.pdf in \install subdirectory

IBM WebSphere Host Publisher Programmer's Guide and Reference
proggd.htm and proggd.pdf in \proggd subdirectory

IBM WebSphere Host Publisher Messages Reference
msgref.htm and msgref.pdf in \msgref subdirectory

IBM Host Publisher *Readme*
readme.htm

Information on the Web

Find the most up-to-date versions of this document, frequently asked questions (FAQs), white papers, and additional information at the product Web site: http://www.ibm.com/software/network/hostpublisher.

Chapter 1. Programming with IBM Host Publisher Integration Objects

Host Publisher Studio creates Java™ objects (Integration Objects, Remote Integration Objects, EJB Access Beans, and Web Services support files) that can be used as building blocks for WebSphere applications. Host Publisher Java objects are Java beans that encapsulate interactions with legacy data sources. These legacy data sources include terminal-oriented applications that use 3270, 5250, and VT data streams, as well as relational databases that provide a JDBC™ interface, such as the IBM DB2 Universal Database®.

The following sections introduce the components of a WebSphere application. This information helps the Host Publisher programmer understand how Host Publisher Java objects can be used in a WebSphere application.

Application components

Application components are those that a developer will have to program, whether manually or with the aid of tools. The language used to develop a given application component will depend in large part upon the "tier" where the component will be executed at runtime.

For example, browser-based components will tend to use tag and script-oriented languages, while Web application server components will tend towards Java.

Because the language differences tend to divide along tier boundaries, the following sections describe the components you develop that are hosted by browsers, Web application servers, and distributed object servers.

Browser components

While a browser is not provided by WebSphere, browser components make up a large part of any Web-enabled application. The reason is that the browser serves as the runtime engine for the user interface of a Web application.

The browser components that are most relevant to the WebSphere programming model include:

- HTML
- DHTML and JavaScript
- Framesets and Named Windows
- eXtensible Markup Language (XML), XML Style Language (XSL) and Document Type Definition (DTD).

Web application server components

Web application server components are different from browser components in that Web application server components can create dynamic pages. HTML, DHTML, and framesets cover the static components of the programming model. The Web application server components hosted by WebSphere most useful in generating dynamic content include:

- Servlets
- JavaServer Pages™ (JSPs).

© Copyright IBM Corp. 2000, 2002

Distributed object server components

The WebSphere programming model provides support for Java-based distributed objects called Enterprise JavaBeans™ (EJBs). EJBs can be thought of as a standard mechanism to contain enterprise business logic and data (usually hosted on some enterprise server) that can take advantage of the following object services:

- Distribution, the ability for the server to be remote from the client
- Persistence, maintenance of the essential data associated with the component
- Transactions, providing the atomicity, consistency, isolation, and durability (ACID) characteristics for the units of work
- Security, control of the roles that can access the objects and associated methods
- Trace and monitoring, configurable instrumentation for debugging and performance tuning.

Using Host Publisher Java objects in a WebSphere application

Refer to Figure 1 for an overview of where Host Publisher Java objects can be used within the "tiers" of a WebSphere application.

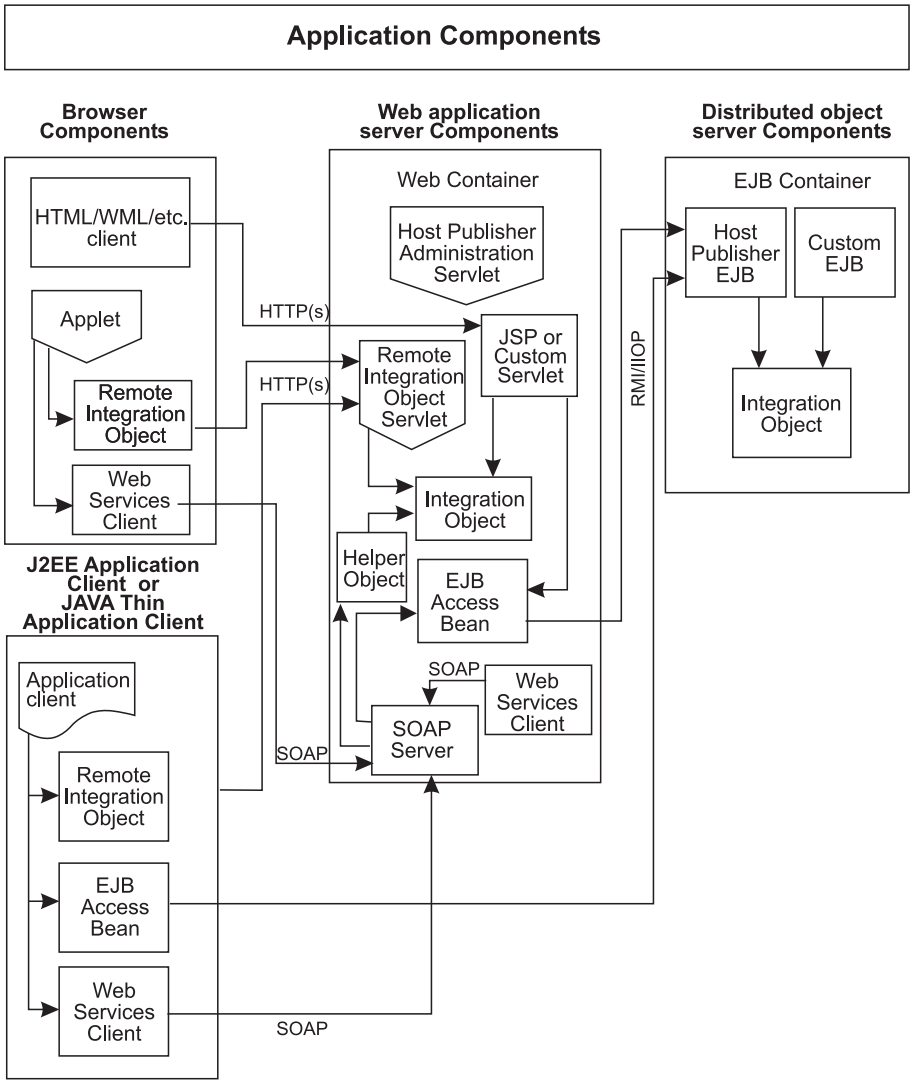


Figure 1. Host Publisher Java objects in a WebSphere application

Host Publisher Server installation configures WebSphere AE to define the Host Publisher application server (named HostPubServer in the WebSphere Administration Console.) In WebSphere AEs, Host Publisher runs in the existing application Server. The installation of Host Publisher Server deploys three enterprise applications in the application server:

HPAdmin.ear
Provides administrative support for Host Publisher Server.

xmlLegacyGW.ear
Provides access to the XML Gateway function.

HPDoc.ear
Provides internationalization support for Host Publisher documentation

Host Publisher Integration Objects are intended to run only where Host Publisher Server is installed. If you want to build an application that will run remotely from Host Publisher Server yet requires the function of an Integration Object, use Web Services, Remote Integration Objects, or EJB Access Beans.

Because Host Publisher Java objects are reusable Java beans, they can be used to build your own custom servlets and EJBs.

You can build other configurations using WebSphere configuration, such as:

- Cloning your Host Publisher application server to provide load balancing and hot standby
- Defining your own application server to run Host Publisher J2EE applications, to isolate the processing of certain applications in one application server from those in another application server.

Refer to WebSphere documentation for information on WebSphere configuration.

The versatility of Host Publisher Java objects in WebSphere applications, combined with the ability to build other WebSphere configurations, enables you to build Web applications to meet your business needs.

Using Host Publisher Integration Objects

You can use the Integration Objects that Host Publisher creates to build your own server-side components, such as servlets, JSPs, or EJBs. This section provides instructions for setting up an integrated development environment (IDE) to work with an Integration Object, sample code to use when you set up the Host Publisher Server runtime environment in an IDE, and instructions for working with Integration Objects in a servlet, JSP, or EJB.

The IDEs described in this chapter are WebSphere Studio tools, such as WebSphere Studio Application Developer (WSAD) Version 4.

Preparing to work with an Integration Object

These instructions assume you have already created an Integration Object using the Host Publisher **Database Access** or **Host Access** application, and that you are developing your program on the machine where Host Publisher Studio is installed.

In these instructions:

- *Studio_Install_Dir* is the Host Publisher Studio installation directory
- *IOName* is the name you gave to the Integration Object.

- *appname* is the name you gave the application in the Host Publisher Studio Application Integrator.

Use Host Publisher Studio Application Integrator to build an Enterprise Archive (.ear) file with the files needed for further development.

You can use your IDE to create servlets, EJBs, or JSPs, which will need access to the following common files located in the *Studio_Install_Dir*\Common directory:

- elf.jar
- habeansnlv.jar
- HpRte.jar
- HPShared.jar
- HPubCommon.jar

  Note: For an EJB Web project, this is the only common file required. If you use the sample code for starting and stopping the Host Publisher Server instance, you also need access to HpRte.jar.
- HPubService.jar
- log.jar
- sslight-ex11-rsa-des.zip
- xmlLegacyPortal.jar

If you run an Integration Object that is express logon enabled, you also need access to the HostPubELF.class file that you created in Host Publisher Studio. See the *IBM WebSphere Host Publisher Administrator's and User's Guide* for information on creating this file.

Working with Host Publisher .ear, .war, and .jar files in WSAD

To work with .ear, .war, and .jar files in WSAD, the following are the general steps required:

1. Create an application using Host Publisher Studio Application Integrator and import the required Integration Objects.

   Note: You can create the JSPs for an application in Host Publisher Studio Application Integrator and enhance them in WSAD, or create the JSPs in WSAD.
2. Click Finish and choose Create J2EE Archives from the toolbar to create the .ear file.
3. Import the .ear, .war, or .jar file located in the *Studio_Install_Dir*\Applications*appname* directory into WSAD. Modify the Java build path for Web projects and EJB .jar projects to include the common files listed in "Preparing to work with an Integration Object" on page 3.
4. Set up your application server instance.
5. Start the Host Publisher runtime. See "Sample code for starting and stopping the Host Publisher Server runtime" on page 5 for the location of sample code.
6. Continue development of your J2EE applications in WSAD.

The following information provides more specific information on the previous general steps. You can import .ear and Web Application Record (.war) files created by Host Publisher Studio into WSAD for testing and access to the tools provided by the integrated development environment (IDE). If you import a .ear file, WSAD creates an Enterprise Application project for the .ear file and a Web project for the

.war file contained in the .ear file. If you import a .war file, you can associate the Web project with either an existing Enterprise Application project or a new Enterprise Application project.

WSAD also provides EJB support. If you import a .ear file that contains an Enterprise JavaBeans (EJB) .jar file, you also have an EJB .jar project. You can import EJB .jar and .war files into separate Enterprise Application projects. You must generate the EJB deployment code in WSAD before running the application.

All of these projects can be built to run on the application server instance. Projects can run within WSAD using the WebSphere Test Environment, or remotelyon an installed WebSphere Application Server using WSAD Remote Server support. See the online help of your IDE for information about setting up the Remote Test Environment or specifying the type of server instance to run your project.

Setting up the WebSphere Test Environment in WSAD

If you plan to use Host Publisher Integration Objects in the WebSphere Test Environment, you must create a servlet to start the Host Publisher Server runtime. Create a project to contain the servlet. See "Sample code for starting and stopping the Host Publisher Server runtime" for the location of sample code.

To build the servlet for starting the Host Publisher Server runtime in the WebSphere Test Environment, your IDE will need access to the common files listed in "Preparing to work with an Integration Object" on page 3. Add these files to the Java build path for the project containing the servlet.

Running the servlet in the WebSphere Test Environment publishes the servlet and runs it in a Web browser. A default server instance and configuration for the WebSphere Test Environment are created, if they do not already exist. An error occurs when you run the servlet on the server for the first time. You need to modify the default server environment that was created for you, by adding the common files listed in "Preparing to work with an Integration Object" on page 3 to the WebSphere specific classpath in the WebSphere Test Environment server instance. Restart the WebSphere Test Environment server instance for the changes to take effect.

Sample code for starting and stopping the Host Publisher Server runtime

You need to start the runtime if your application is running in a WebSphere application server other than the Host Publisher application server. The runtime should be started before any Integration Objects are executed. One way to accomplish this is to define a servlet containing code to start and stop the runtime.

Sample source code for starting and stopping the runtime and all documentation can be accessed starting with the following file:

Studio_Install_Dir\SDK\Server\Introduction.html

Using an Integration Object in a Web container (custom servlet or JSP)

The instructions in this section reference Integration Object methods. See "Integration Object methods" on page 11 for a description of these methods.

Note: You must set your CLASSPATH so that your servlet can access your Integration Object's .jar file as well as the common .jar files listed in "Preparing to work with an Integration Object" on page 3.

To write a servlet that invokes an Integration Object:

1. If your servlet will run in a WebSphere application server other than the one configured during the installation of Host Publisher Server, you must initalize and start the server. You can start the server with a custom service or by creating a servlet to start the server. See "Defining Host Publisher Server as a WebSphere custom service" on page 10 for information on using custom services. See "Sample code for starting and stopping the Host Publisher Server runtime" on page 5 for the location of sample code. In the WebSphere Test Environment, you must initialize and start the server using a servlet.
2. Create an instance of your Integration Object by calling its constructor.
3. Invoke the methods for the Integration Object. You can invoke methods to set properties of input variables. The naming convention for setter methods is as follows:

   ```
   void setXyz(String)
   ```

   where *xyz* is the name of your input variable.

   You can use a different connection pool than the one you specified when you created your Integration Object. To specify a different connection pool, invoke the method

   ```
   void setHPubStartPoolName(String)
   ```

   specifying the name of the connection pool you want to use.

   Note: You can add additional connection pools to the .ear file built in Host Publisher Studio Application Integrator.
4. Invoke the Integration Object to perform its task (running a macro or querying a database, for example):

   ```
   void doHPTransaction(HttpServletRequest, HttpServletResponse)
   ```
5. Check for errors. The doHPTransaction(HttpServletRequest, HttpServletResponse) method will throw an exception (of type com.ibm.HostPublisher.IntegrationObject.BeanException) if the Integration Object has an error.

   When the Integration Object is called by a JSP, the JSP processor will catch the exception and redirect the browser to the error page specified on the `errorPage="errorPageName"` attribute of the `page` directive. Refer to the Host Publisher default error page, DefaultErrorPage.jsp, which can be found in the *`Studio_Install_Dir`*`\Studio` directory, for a sample.

   When the Integration Object is called by a custom servlet, your code must catch the thrown exception:

   ```
   try {
        integrationObject.doHPTransaction(request, response);
       } catch (Exception e) {
       // Handle the exception condition and recover
       }
   ```

   Note: The error handling method used in previous versions of Host Publisher, which used servlet response redirection to an error page specified by a call to setHPubErrorPage(String), is deprecated. It will work for Integration Objects created with previous versions of Host Publisher, but cannot be used for Integration Objects created with Host Publisher 4.0.
6. Request the results from your Integration Object.

- If you created your Integration Object using the **Host Access** application, retrieve the value for output variables by invoking one of the following methods:
  - Simple text

    ```
    String getAbc()
    ```

    where abc is the name of your output variable.
  - Tables
    - To get an entire column of results, invoke

      ```
      String[] getAbc()
      ```

      where abc is the name of your output variable.
    - To get a single value from a column of results, invoke

      ```
      String getAbc(int) throws ArrayIndexOutOfBoundsException
      ```

      where abc is the name of your output variable, and int is the index of the value you want. As you iterate through the array, the method will throw an ArrayIndexOutOfBoundsException exception when you have reached the end of the array.
- If you created your Integration Object using the **Database Access** application and your Statement Type was **Select** or **Select Unique**, your output variables are the columns of the table you queried.
  - To get an entire column of results, invoke

    ```
    String[] getTableColumn_()
    ```

    where Table is the name of the database table and Column is the name of the column in the table.
  - To get a single value from a column of results, invoke

    ```
    String getTableColumn_(int) throws ArrayIndexOutOfBoundsException
    ```

    where Table is the name of the database table, Column is the name of the column in the table, and int is the index of the value you want. As you iterate through the array, the method will throw an ArrayIndexOutOfBoundsException exception when you have reached the end of the array.
- If you created your Integration Object using the **Database Access** application and your Statement Type was **Insert**, **Update**, or **Delete**, you have only one output variable. To get the number of rows changed by your database request, invoke the method

  ```
  int getHPubNumberOfRowsChanged()
  ```
- Regardless of the application you used to create your Integration Object, you can invoke the XML method

  ```
  String getHPubXMLProperties()
  ```

  which returns the IntegrationObject's properties and values as an XML formatted string.

The input variables for all Integration Objects have getter methods corresponding to each setter method so that you may retrieve those values if necessary. The signature for these methods is

```
void getXyz(String)
```

where xyz is the name of your input variable.

To verify input or output variable names that are generated from data that you entered, look at the properties defined in your Integration Object's BeanInfo java file. The Integration Object's BeanInfo java file is found in the *Studio_Install_Dir*\Studio\IntegrationObjects\ directory.

Using an Integration Object in an EJB container (from a custom EJB)

The instructions in this section reference Integration Object methods. See "Integration Object methods" on page 11 for a description of these methods.

To use an Integration Object from a custom EJB:

1. If your custom EJB will run in a WebSphere application server other than the one configured during the installation of Host Publisher Server, you must initalize and start the server. You can start the server with a custom service or by creating a servlet to start the server. See "Defining Host Publisher Server as a WebSphere custom service" on page 10 for information on using custom services. See "Sample code for starting and stopping the Host Publisher Server runtime" on page 5 for the location of sample code. In the WebSphere Test Environment, you must initialize and start the server using a servlet.
2. Create an instance of your Integration Object by calling its constructor.
3. Invoke the methods for the Integration Object instance. You might want to invoke methods to set properties of input variables. The naming convention for setter methods is as follows:

   ```
   void setXyz(String)
   ```

   where *xyz* is the name of your input variable.

   You can use a different connection than the one you specified when you created your Integration Object. To specify a different connection pool, invoke the method

   ```
   void setHPubStartPoolName(String)
   ```

   and specify the name of the connection you want to use.
4. Invoke the Integration Object to perform its task (running a macro or querying a database, for example), using the method

   ```
   void processRequest() throws BeanException
   ```

   The processRequest() method will throw an exception (of type com.ibm.HostPublisher.IntegrationObject.BeanException) if the Integration Object has an error.

   You can reset the input variables and invoke the processRequest() method multiple times. The error indications and result values will be reset with each invocation.
5. Check for errors by invoking

   ```
   int getHPubErrorOccurred()
   ```

   If your result is nonzero, an error has occurred. You will have an error exception and, for **Database Access** Integration Objects, you might have an SQL error exception. To get the specific exception for the error, invoke

   ```
   Exception getHPubErrorException()
   ```

You can retrieve the error message by invoking getMessage() on the Exception object. The messages are documented in the *IBM WebSphere Host Publisher Messages Reference*. Note that the first seven characters are set to HPSxxxx where xxxx is the message number.

For **Database Access** Integration Objects with a nonzero result from getHPubErrorOccurred(), check to see whether the error message number is in the range 6205-6209. If so, you have an SQL error exception. In the case where your database action caused more than one SQL error to be generated, you are returned the first SQL error. To get the first SQL error exception, invoke

```
SQLException getHPubSQLErrorException()
```

In addition to SQL error exceptions, if you created your Integration Object using **Database Access** application, you may have an SQL warning exception. To check for an SQL warning exception, invoke

```
int getHPubWarningOccurred()
```

If your result is nonzero, an SQL warning has occurred. Note that you may have an SQL error exception as well as an SQL warning exception. In the case where your database action caused more than one SQL warning to be generated, you are returned the first warning generated. To get the first warning exception, invoke

```
SQLWarning getHPubSQLWarningException()
```

6. Request the results from your Integration Object.
   - If you created your Integration Object using the **Host Access** application, retrieve the value for output variables by invoking one of the following methods:
     - Simple text

       ```
       String getAbc()
       ```

       where abc is the name of your output variable.
     - Tables
       - To get an entire column of results

         ```
         String[] getAbc()
         ```

         where abc is the name of your output variable.
       - To get a single value from a column of results

         ```
         String getAbc(int) throws ArrayIndexOutOfBoundsException
         ```

         where abc is the name of your output variable, and int is the index of the value you want. As you iterate through the array, the method will throw an ArrayIndexOutOfBoundsException exception when you have reached the end of the array.
   - If you created your Integration Object using the **Database Access** application and your Statement Type was **Select** or **Select Unique**, your output variables are the columns of the table you queried. Retrieve the value for output variables by invoking one of the following methods:
     - To get an entire column of results

       ```
       String[] getTableColumn_()
       ```

       where Table is the name of the database table and Column is the name of the column in the table.
     - To get a single value from a column of results

```
String getTableColumn_(int) throws ArrayIndexOutOfBoundsException
```

where Table is the name of the database table, Column is the name of the column in the table, and int is the index of the value you want. As you iterate through the array, the method will throw an ArrayIndexOutOfBoundsException exception when you have reached the end of the array.

- If you created your Integration Object using the **Database Access** application and your Statement Type was **Insert**, **Update**, or **Delete**, you have only one output variable. To get the number of rows changed by your database request, invoke the method

  ```
  int getHPubNumberOfRowsChanged()
  ```

- Regardless of the application you used to create your Integration Object, you can invoke the XML method

  ```
  String getHPubXMLProperties()
  ```

  which returns the IntegrationObject's properties and values as an XML formatted string.

The input variables for all Integration Objects have getter methods corresponding to each setter method so that you can retrieve those values if necessary. The signature for these methods is

```
void getXyz(String)
```

where xyz is the name of your input variable.

If you are unsure about any input or output variable names that are generated from data that you entered, look at the properties defined in your Integration Object's BeanInfo java file. The Integration Object's BeanInfo java file is found in the *Studio_Install_Dir*\Studio\IntegrationObjects\ directory.

Defining Host Publisher Server as a WebSphere custom service

To run Host Publisher applications in a WebSphere application server, the Host Publisher Server runtime must be initialized and active. The most efficient way to address this requirement is to define the Host Publisher Server runtime as a custom service associated with the application server. A custom service is a "hook point" defined by WebSphere that gets executed at application server startup and shutdown. By defining the Host Publisher Server runtime as a custom service, the runtime is automatically initialized and started at application server startup and is terminated when the application server is stopped.

For information on creating a custom service, refer to WebSphere Application Server documentation.

When creating a custom service for Host Publisher, the following properties should be defined:

- Enable the custom service.
- Provide the class name as *com.ibm.HostPublisher.Server.StartRTE*.
- Provide a unique name on WebSphere Application Server AE or a unique display name on WebSphere Application Server AEs.
- For WebSphere Application Server AE, define a custom property named *hostpublisher.install.dir* with a value of the installation directory of Host Publisher

Server. For WebSphere Application Server AEs, define a dynamic property named *hostpublisher.install.dir* with a value of the installation directory of Host Publisher Server.

Integration Object methods

Host Publisher Integration Objects contain Java methods that you can use when programming with Integration Objects. Some of the methods apply only to Host Access Integration Objects and some of the methods apply only to Database Access Integration Objects. This section lists the methods and a short description of the function of each method.

Common methods

These methods are common to all Integration Objects:

void doHPTransaction(HttpServletRequest req, HttpServletResponse resp) throws BeanException

This execution method runs a Host Publisher Integration Object or EJB Access Bean from a servlet or JSP. If you use this method, you can run chained Integration Objects without additional programming to chain Integration Objects.

void processRequest() throws BeanException

This execution method runs a Host Publisher Integration Object, Remote Integration Object, or EJB Access Bean from an application component. This method does not require the application component to be running in a Web container. To run chained Integration Objects, you have to do additional programming; refer to "Integration Object chaining" on page 13.

java.lang.String getHPubBeanName()

Returns the name of the current Integration Object, EJB Access Bean, or Remote Integration Object

java.lang.String getHPubBeanType()

Returns a string representing the type of Host Publisher Integration Object, EJB Access Bean, or Remote Integration Object. The returned string can be one of the following:

HOD The bean was created using Host Access.

DB This bean was created using Database Access.

void setHPubErrorPage(java.lang.String value)

For Integration Objects created with previous versions of Host Publisher, sets the name of the error page to be used. Use this method only if you are running the Host Publisher Integration Object or EJB Access Bean from a servlet or JSP. Specify the name of your error page relative to the location of your servlet or JSP.

Note: This method is deprecated and cannot be used for Integration Objects created with Host Publisher 4.0.

java.lang.String getHPubStartPoolName()

Returns the name of the connection pool from which the Integration Object acquired the connection

void setHPubStartPoolName(java.lang.String value)

Sets the name of the connection pool from which the Integration Object will acquire the connection

java.lang.String getHPubXMLProperties()
Returns an XML formatted string specifying the property names and values for this Integration Object

java.lang.String getHPubXMLProperties(HPubConvertToTableFormat.xsl)
Returns an XML formatted string specifying the property names and values for this Integration Object, and applies XML stylesheet processing to the returned string. See "Chapter 2. Applying XML stylesheet processing to Integration Object output" on page 15 for more information.

int getHPubErrorOccurred()
Returns a non-zero value when an error has occurred.

java.lang.Exception getHPubErrorException()
Returns an exception object that describes the error that occurred; valid only if HPubErrorOccurred is non-zero

java.lang.String getHPubErrorMessage()
Returns a string containing the Host Publisher code and message of the error that occurred; valid only if HPubErrorOccurred is non-zero

Host Access Integration Object methods

These methods are unique to Host Access Integration Objects:

java.lang.String getHPubLinkKey()
Returns the name of the key that represents the connection for the Integration Object chain. This value should be obtained from the first Integration Object in a chain after the Integration Object has run in a non-Web container.

void setHPubLinkKey(java.lang.String value)
Sets the name of the key that represents the connection for the Integration Object chain. This value should be set for any chained Integration Objects, other than the first Integration Object in the chain, before they run in a non-Web container.

java.lang.String getHPubStartChainName()
Returns the name of the start state label as defined in Host Access. This value is *Null* for the first Integration Object in a chain or an Integration Object that is not chained.

java.lang.String getHPubEndChainName()
Returns the stop state label as defined in Host Access. This value is *Null* for the last Integration Object in a chain or an Integration Object that is not chained.

java.lang.String getHPubScreenState()
Returns the name of the last Host On-Demand macro screen executed when the macro was stopped.

java.lang.String getHPubMacroMessage()
Returns the value of the message tag of the last screen executed in the current Host On-Demand macro screen. See "<message> tag" on page 77 for information about using this method for debugging Host Publisher macro execution.

Database Access Integration Object methods

These methods are unique to Database Access Integration Objects:

int getHPubNumberOfRowsChanged()
Returns the number of rows changed by this database request

java.lang.String getHPubWarningOccurred()
Returns a non-zero value indicating that a warning has occurred.

java.sql.SQLWarning getHPubSQLWarningException()
Returns a SQLWarning object of the warning that occurred; valid only if HPubWarningOccurred is non-zero

java.sql.SQLException getHPubSQLErrorException()
Returns a SQL Exception object of the error that occurred; valid only if HPubErrorOccurred is non-zero and HPubErrorMessage indicates an SQL error

Integration Object chaining

Integration Object chaining is handled by Host Publisher for the following:

- Host Publisher applications using Host Access Integration Objects or the corresponding EJB Access Beans
- Custom JSPs or servlets that use Host Access Integration Objects or the corresponding EJB Access Beans in a Web container.

In these cases, the *doHPTransaction* execution method is used.

You must retrieve and set properties that enable chaining for the following:

- Remote Integration Objects
- EJB Access Beans running outside of a Web container
- Custom EJBs that use Integration Objects.

In these cases, the *processRequest* execution method is used.

See "Integration Object methods" on page 11 for a description of the *doHPTransaction* and *processRequest* methods.

Host Publisher provides methods that enable you to extract the key that represents the connection for the Integration Object chain from the first Integration Object in a chain, and to set the property for subsequent Integration Objects in the chain.

You must also retrieve and set properties that enable chaining for Web Services. For Web Services, the *processWSRequest* execution method is used. See "Programming with Web Services Integration Objects and EJB Access Beans" on page 27 for a description of the *processWSRequest* method.

For a description of Integration Object chaining for Remote Integration Objects, see "Remote Integration Object chaining" on page 33.

For a description of Integration Object chaining for EJB Access Beans outside of a Web container, see "EJB Access Bean chaining" on page 23.

To build an Integration Object chain from a custom EJB, do the following:

1. If your custom EJB is deployed in a container other than the one configured in the Host Publisher application server, initialize and start the server. You can start the server with a custom service or by creating a servlet to start the server. See "Defining Host Publisher Server as a WebSphere custom service" on page 10 for information on using custom services. See "Sample code for starting and stopping the Host Publisher Server runtime" on page 5 for the location of sample code.
2. Create an instance of the first Integration Object in the chain by calling its constructor.

3. Invoke the methods for the Integration Object instance. You might want to invoke methods to set properties of input variables. The naming convention for setter methods is as follows:

   ```
   IOChain1.setXyz(String)
   ```

   where *xyz* is the name of your input variable.
4. Invoke the Integration Object to perform its task (running a macro or querying a database, for example), using the method

   ```
   IOChain1.processRequest()
   ```
5. Check for errors by invoking

   ```
   IOChain1.getHPubErrorOccurred()
   ```
6. Extract and save the key that represents the connection for the Integration Object chain

   ```
   String myLinkkey = IOChain1.getHpubLinkKey();
   ```
7. Create an instance of the next Integration Object in the chain by calling its constructor.
8. Set the key for this chained connection

   ```
   IOChain2.setHpubLinkKey(myLinkkey);
   ```
9. Invoke the methods for this Integration Object instance. You might want to invoke methods to set properties of input variables. The naming convention for setter methods is as follows:

   ```
   IOChain2.setXyz(String)
   ```

   where *xyz* is the name of your input variable.
10. Invoke this Integration Object to perform its task (running a macro or querying a database, for example), using the method

    ```
    IOChain2.processRequest()
    ```
11. Check for errors by invoking

    ```
    IOChain2.getHPubErrorOccurred()
    ```

Repeat steps 7 through 11 for any and all subsequent Integration Objects in the chain.

Integration Object chaining and WebSphere cloning

If you are using WebSphere for cloning application servers for load balancing, and you are using Integration Objects chaining in a servlet or JSP, you must configure HTTP session affinity to ensure that the same Web client (browser) returns to the WebSphere application server that originally established the host connection.

Chapter 2. Applying XML stylesheet processing to Integration Object output

With Host Publisher Version 2, Integration Object data in XML format was retrieved using the getHPubXMLProperties() function. XML stylesheets could only be applied to Remote Integration Object XML output.

Host Publisher now provides an XML stylesheet, HPubConvertToTableFormat.xsl, that can be applied to the getHPubXMLProperties() function call for tabular data. Applying the stylesheet produces an XML format including the table name and column names, and reorders data in record format. To apply the HPubConvertToTableFormat.xsl stylesheet, you must code the getHPubXMLProperties() function call as getHPubXMLProperties(HPubConvertToTableFormat.xsl).

For information on the methods that you can use in WebSphere applications, see "Integration Object methods" on page 11.

DTD of XML data returned by getHPubXMLProperties() method

When an XML stylesheet is not applied to Integration Object output, the XML data is returned with the following document type definition (DTD):

```
<?xml version=\"1.0\" standalone=\"yes\"?>
<!DOCTYPE com.ibm.HostPublisher.IntegrationObject.properties [
<!ELEMENT com.ibm.HostPublisher.IntegrationObject.properties
             (inputProperties, outputProperties)>
<!ATTLIST com.ibm.HostPublisher.IntegrationObject.properties name CDATA "">
<!ELEMENT inputProperties (inputProperty*)>
<!ELEMENT inputProperty (value)>
<!ATTLIST inputProperty name CDATA "">
<!ELEMENT outputProperties (outputProperty*)>
<!ELEMENT outputProperty (value*)>
<!ATTLIST outputProperty name CDATA "" type (singlevalue|multivalue) 'multivalue'>
<!ELEMENT value (#PCDATA)>
]>
```

XML data using the getHPubXMLProperties() method

The following sample data:

Table 1. Sample XML data

Name	Phone Number
Mary Smith	867–5309
John Doe	123–4567

results in the following XML data:

```
<com.ibm.HostPublisher.IntegrationObject.properties name=IntegrationObject.test1>
<inputProperties>
<inputProperty name=nameValue>
<value>%</value>
<inputProperty>
</inputProperties?
<outputProperties>
<outputProperty name=tablename type=multivalue>
<value>Mary Smith</value>
<value>John Doe</value>
```

© Copyright IBM Corp. 2000, 2002

```
</outputProperty>
<outputProperty name=table1phonenumber type=multivalue>
<value>867-5309</value>
<value>123-4567</value>
</outputProperty>
<outputProperty name=databaseStatus type=singlevalue>
<value>Ok</value>
</outputProperty>
<outputProperty name=hPubErrorOccurred" type=singlevalue>
<value>0</value>
</outputProperty>
<outputProperty name=hPubErrorException" type=singlevalue>
<value></value>
</outputProperty>
<outputProperty name=hPubErrorMessage" type=singlevalue>
<value></value>
</outputProperty>
</outputProperties>
</com.ibm.HostPublisher.IntegrationObject.properties>
```

All of the data is within multiple <value> tags with the <outputProperty> tags in columnar order.

DTD of XML data returned by getHPubXMLProperties(HPubConvertToTableFormat.xsl) method

When the XML HPubConvertToTableFormat stylesheet is applied to Integration Object output, the XML data is returned with the following document type definition (DTD):

```
<?xml version=\"1.0\" standalone=\"yes\"?>
<!DOCTYPE com.ibm.HostPublisher.IntegrationObject.properties [
<!ELEMENT com.ibm.HostPublisher.IntegrationObject.properties
            (inputProperties, outputProperties)>
<!ATTLIST com.ibm.HostPublisher.IntegrationObject.properties name CDATA "">
<!ELEMENT inputProperties (inputProperty*)>
<!ATTLIST inputProperty name CDATA "">
<!ELEMENT outputProperties (outputProperty*,Table*)>
<!ATTLIST outputProperty name CDATA "">
<!ELEMENT Table (DataRecord*)>
<!ATTLIST Table name CDATA "">
<!ELEMENT DataRecord (outputProperty*)>
]>
```

XML data with HPubConvertToTableFormat stylesheet applied

The sample data shown in Table 1 on page 15 results in the following XML data:

```
<com.ibm.HostPublisher.IntegrationObject.properties name=IntegrationObject.test1>
<!-- Input Properties -->
<inputProperties>
<inputProperty name=inputName>
%
<inputProperty>
</inputProperties?
<outputProperties>

<!-- Table (multivalued) output property -->
<Table name=table1>
<DataRecord>
<outputProperty name=Name>Mary Smith</outputProperty>
<outputProperty name=phoneNumber>867-5309</outputProperty>
</DataRecord>
<DataRecord>
<outputProperty name=Name>John Doe</outputProperty>
<outputProperty name=phoneNumber>123-4567</outputProperty>
```

```
</DataRecord>
</Table>

<!-- Single Valued output Property -->
<outputProperty name=databaseStatus>
Ok
</outputProperty>

<!-- Standard Error  output Properties -->
<outputProperty name=hPubErrorOccurred">0</outputProperty>
<outputProperty name=hPubErrorException"></outputProperty>
<outputProperty name=hPubErrorMessage"></outputProperty>
</outputProperties>

</com.ibm.HostPublisher.IntegrationObject.properties>
```

Chapter 3. Programming with the XML Java bean

The Host Publisher XML Java bean presents the view of a host application in XML format, showing one screen at a time. You can write a Java application, applet, Java bean, or servlet to convert the XML data to HTML or some other markup language.

Host Publisher XML Gateway SDK includes the following:

- The xmlAppData Java bean
- The sample xmlGateway servlet
- The HostConnection Java bean

The xmlAppData Java bean encapsulates the host application XML data, the sample xmlGateway servlet converts the XML data to HTML format, and the HostConnection Java bean acquires and releases Host On-Demand session Java beans using the Host Publisher Server runtime environment.

See Figure 2 for an architectural overview of the Host Publisher XML Gateway.

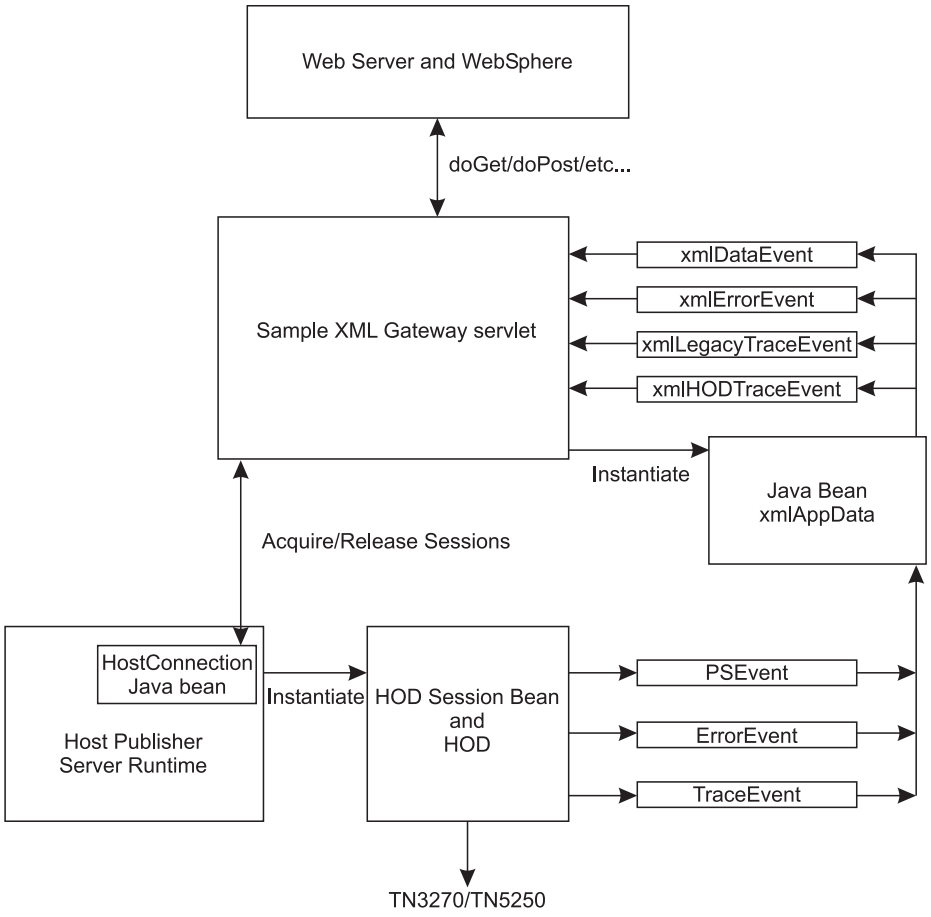


Figure 2. XML Gateway Architecture

© Copyright IBM Corp. 2000, 2002

The xmlAppData Java bean

This Java bean communicates with the host application using XML data formatting. Javadoc for this Java bean is included with the HostPublisher installation and can be accessed starting with the following file:

install_dir\SDK\XLGW\xmlAppData\AllNames.html

where *install_dir* is the directory in which Host Publisher is installed.

Refer to Figure 2 to see how the xmlAppData Java bean relates to the other components of the XML Gateway.

The xmlAppData Java bean contains properties that describe the HostPublisher, HOD, and TN3270 or TN5250 parameters used to instantiate a host session.

The xmlAppData Java bean also contains properties that describe the host data as XML records. These records contain the text of the fields displayed on the screen and the attributes of the fields. Methods are supplied for reading the fields so they can be manipulated as an XML document.

The xmlAppData Java bean contains methods for sending a new screen of data to the host. This new screen is described by an XML document, which will often be the previous host data transformed into an XML document, with the user input fields updated, or a user action specified, or both.

The xmlAppData Java bean sends xmlDataEvents to all xmlDataEvent listeners. The data events are sent when the host screen has changed.

The xmlAppData Java bean interacts with the IBM Host On-Demand (HOD) session Java bean using PSEvents. When a PSEvent is received by the xmlAppData Java bean, the Java bean updates its internal objects to reflect the new status of the host screen. The xmlAppData Java bean sends an xmlDataEvent to inform its listeners of the change.

The xmlAppData Java bean sends xmlErrorEvents to all xmlErrorEvent listeners when a processing error has been detected.

The xmlAppData Java bean sends xmlLegacyTraceEvents to all xmlLegacyTraceEvent listeners for appropriate tracing of the xmlAppData Java bean activity.

The xmlAppData Java bean listens for HODTraceEvents and sends the HODTraceEvents to all xmlHODTraceEvent listeners.

The sample XML Gateway servlet

The XML Gateway servlet transforms XML data into HTML format. This servlet enables viewing of a host screen in a Web browser. A user can interact with the host screen in the browser by typing in the fields on the screen or by using the function keys, which are displayed as buttons in the browser. Because the servlet interacts with the host application, most of the data processing takes place at the server. This enables browser access to host applications from a thin client.

You can write your own servlet to interface with the host application. To facilitate the writing of Host Publisher XML Gateway servlets, the sample source code for

the XML Gateway servlet is included with Host Publisher. HTML documentation is also included. The source code and all documentation can be accessed starting with the following file:

install_dir\SDK\XLGW\Introduction.html

where *install_dir* is the directory in which Host Publisher is installed.

The sample servlet uses Extensible Stylesheet Language (XSL) processing to transform XML data to HTML format. This is a powerful example of how to transform the host screen into XML data and, through the use of a stylesheet, present it to the end user in a different format. By replacing or modifying the sample servlet, the supplied stylesheet, or both, and by using an XSL processor like the one included in WebSphere Application Server, the application programmer can easily render host data onto a variety of devices using a single servlet with multiple stylesheets.

Refer to Figure 2 to see how the XML Gateway servlet relates to the other components of the XML Gateway.

The XML Gateway servlet:

1. Is instantiated with form parameters that describe the desired host session. These parameters describe the telnet name of the host, the terminal format (3270 or 5250) of the session, and so on.
2. Initiates a host session using Host Publisher and IBM Host On-Demand Java objects and properties.
3. Instantiates the xmlAppData Java bean and makes the Java bean a listener for Host On-Demand events.
4. Retrieves, after a programmed delay, the current state of the host screen as XML data from the xmlAppData Java bean.
5. Formats the XML data as HTML output. This output is returned to the browser.

The HTML output returned to the browser shows how the screen displays on a traditional terminal. The user can input data directly onto the HTML host screen. The user can move the cursor to the input fields using the mouse or the **Tab** key. The traditional terminal function keys are presented to the user as buttons on the browser screen. The user can select the buttons using the mouse or using the **Tab** key and pressing the **Enter** key on the keyboard.

Two additional buttons are presented on the user's browser page: **Refresh** and **Disconnect**. The **Refresh** button updates the browser's host screen to the current state of the host session, ignoring possible input. The **Disconnect** button terminates the current host session. Disconnection enables an efficient use of Host Publisher resources. The user should disconnect the host session when interaction with the host application is no longer needed.

While the servlet is a useful application, it is an example of how to interact with Host Publisher to encapsulate host data in XML format. The servlet could be changed to interact with the host application using XML processing techniques, in an automated fashion, presenting the user with a specific subset of information obtained from the host.

The data can also be rendered in different formats by using a different XSL stylesheet when processing the data. The sample servlet can be changed to render the data in a format that matches the output preference of the user or the user's

access device. The sample servlet can do this at runtime by specifying the XSL stylesheet used for this particular instance of the servlet.

The HostConnection Java bean

This Java bean is part of the Host Publisher Server runtime code. Javadoc for this Java bean is included with the HostPublisher installation and can be accessed starting with the following file:

install_dir\SDK\XLGW\HostConnection\AllNames.html

where *install_dir* is the directory in which Host Publisher is installed.

The HostConnection Java bean contains methods for acquiring and releasing Host On-Demand session Java beans using the Host Publisher Server runtime environment.

Chapter 4. Using Enterprise JavaBeans Support

When you create Enterprise JavaBeans (EJB) Integration Object support in Host Publisher Studio with Host Access or Database Access, EJB support files are generated during creation of the Integration Object. One of the files generated is the EJB Access Bean. The EJB Access Bean is a Java bean that has a signature similar to the Integration Object that the Access Bean supports. You use Host Publisher Studio Application Integrator to import EJB Access Beans to create EJB-based applications to run Integration Objects in an EJB environment. When you choose Create J2EE Archives from the toolbar to create the .ear file, an EJB .jar file is also created, which contains the necessary EJB files.

Programming with EJB Access Beans

Host Publisher Integration Objects contain Java methods that you can use when programming with Integration Objects. These methods are also available when programming with EJB Access Beans. In addition to the methods described in "Integration Object methods" on page 11, you can use the following methods when programming with EJB Access Beans:

java.lang.Object getHPubAccessHandle()
Returns the handle for the Host Publisher EJB instance corresponding to the Integration Object chain

void setHPubAccessHandle(java.lang.Object value)
Sets the handle for the Host Publisher EJB instance corresponding to the Integration Object chain

EJB Access Bean chaining

To support chained Integration Objects, the Host Publisher EJB is implemented as a stateful session bean, where the life cycle of a Host Publisher EJB instance corresponds with the processing of an Integration Object chain. When EJB Access Beans for an Integration Object chain are processed, two EJB Access Bean properties are used:

hPubAccessHandle
Contains the handle for the Host Publisher EJB instance corresponding to the Integration Object chain

hPubLinkKey
Contains the key that represents the connection for the Integration Object chain

EJB Access Bean chaining in a Web container

When EJB Access Beans are processed in a Web container, the EJB Access Bean saves the values of the hPubAccessHandle and hPubLinkKey properties in the HttpSessionObject associated with the client session. The properties are retrieved by subsequent EJB Access Beans in processing the Integration Object chain.

The EJB Access Bean for the first Integration Object in a chain creates an instance of the Host Publisher EJB. That instance is used for all EJB Access Beans for the subsequent Integration Objects in the chain. After the processing of the EJB Access Bean for the last Integration Object in a chain, the Host Publisher EJB instance corresponding to that Integration Object chain is removed.

© Copyright IBM Corp. 2000, 2002

EJB Access Bean chaining outside of a Web container

When EJB Access Beans are processed outside of a Web container (for example, directly from a Java program), the calling program must retrieve the values of the hPubAccessHandle and hPubLinkKey properties. The properties can be retrieved after the EJB Access Bean for the first Integration Object in the chain is processed. The calling program must set the two properties for all subsequent EJB Access Beans in the Integration Object chain before they are processed.

To chain EJB Access Beans outside of a Web container, do the following:

1. Create an instance of the first EJB Access Bean in the chain by calling its constructor.
2. Invoke the methods for the EJB Access Bean instance. You might want to invoke methods to set properties of input variables. The naming convention for setter methods is as follows:

   ```
   EJBChain1.setXyz(String)
   ```

   where *xyz* is the name of your input variable.
3. Invoke the EJB Access Bean to perform its task (running a macro or querying a database, for example), using the method

   ```
   EJBChain1.processRequest()
   ```
4. Check for errors by invoking

   ```
   EJBChain1.getHPubErrorOccurred()
   ```
5. Extract and save the key that represents the connection for the EJB Access Bean chain

   ```
   String myLinkkey = EJBChain1.getHPubLinkKey();
   ```
6. Extract and save the handle for the Host Publisher EJB instance corresponding to the Integration Object chain

   ```
   String myHandle = EJBChain1.getHPubAccessHandle();
   ```
7. Create an instance of the next EJB Access Bean in the chain by calling its constructor.
8. Set the key for this chained connection

   ```
   EJBChain2.setHPubLinkKey(myLinkkey);
   ```
9. Set the handle for this chained connection

   ```
   EJBChain2.setHPubAccessHandle(myHandle);
   ```
10. Invoke the methods for this EJB Access Bean instance. You might want to invoke methods to set properties of input variables. The naming convention for setter methods is as follows:

    ```
    EJBChain2.setXyz(String)
    ```

    where *xyz* is the name of your input variable.
11. Invoke this EJB Access Bean to perform its task (running a macro or querying a database, for example), using the method

    ```
    EJBChain2.processRequest()
    ```
12. Check for errors by invoking

    ```
    EJBChain2.getHPubErrorOccurred()
    ```

Repeat steps 7 through 12 for any and all subsequent EJB Access Beans in the chain.

Using EJB Access Beans with Java Application Clients

With WebSphere 4.0, Java application clients consist of the following models:

- J2EE application client
- Java thin application client

For more information, refer to the WebSphere InfoCenter documentation.

To build Java Application Clients that use EJB Access Beans, the EJB jar file built by the Host Publisher Studio Application Integrator must be deployed to the WebSphere Application Server. This deployment created the EJB stub files required by the EJB Access Beans with the corresponding EJB.

When creating Java Application Clients that use EJB Access Beans, you need the following files:

the deployed EJB .jar file
WebSphere_Install_Dir\installedApps*app_name*

the EJB AccessBean .jar file
Studio_Install_Dir\Studio\Applications*app_name**app_name*.war, where *app_name* is the name of your application. The EJB AccessBean.jar file is in the .war module.

HPubCommon.jar
Studio_Install_Dir\Common\HPubCommon.jar

Chapter 5. Using Web Services support

When you create Web Services Integration Object support in Host Publisher Studio with Host Access or Database Access, Web Services support files are generated during creation of the Integration Object.

The Web Services support files consist of:

- A properties object (named *IONamePropertiesObjectSuffix*),where *IOName* is name of the Integration Object and *PropertiesObjectSuffix* is the name specified for the Web Services Properties Object Suffix. The properties object contains all possible input and output properties for the Integration Object.
- A helper object (named *IONameHelperObjectSuffix*), where *IOName* is the name of the Integration Object and *HelperObjectSuffix* is the name specified for the Web Services Helper Object Suffix. This helper object extends the Integration Object. It initializes the input properties using an instance of the properties object, invokes the Integration Object with the specified input properties, and then returns the output properties via a new instance of the properties object. If you plan to create an Integration Object Web Service within a web application, you will use the helper object to create the Web Service using WebSphere Studio tools

If you want to create Web Services using Host Publisher EJB Access beans, you must create EJB 1.1 Integration Object support, which generates Web Services support files. Use the EJB Access Bean to create the Web Service using WebSphere Studio tools.

Programming with Web Services Integration Objects and EJB Access Beans

Host Publisher Web Services support differs from programming with Integration Objects because only the following method is used to create the Web Service:

IONamePropertiesObjectSuffix **processWSRequest(***IONamePropertiesObjectSuffix***) throws BeanException**

The *processWSRequest* method is contained in the helper object when **Create Web Services support** is checked on the Host Access or Database Access Options menu.The *IONamePropertiesObjectSuffix* object contains the inputs and outputs of an Integration Object. This object is passed to and from the Web Services *processWSRequest* method. It contains getter and setter methods, but no additional methods.

IOName is the name you gave to the Integration Object, and *PropertiesObjectSuffix* is the name specified for the Web Services Properties Object Suffix.

The processWSRequest method takes the properties object as input, drives the Integration Object with those input properties, and returns the output properties of the Integration Object through a new instance of the properties object. The advantage of using the processWSRequest method is that it provides a single method that sets all inputs, drives the Integration Object, and returns all outputs.

© Copyright IBM Corp. 2000, 2002

Integration Object Chaining with Web Services

If your application requires chaining, you must retrieve the hPubLinkKey property from the first Integration Object in the chain, and set it for all subsequent Integration Objects in the chain.

EJB Access Bean Chaining with Web Services

If your application requires chaining, you must retrieve both the hPubLinkKey and the hPubAccessHandle properties from the first EJB Access Bean in the chain, and set them for all subsequent EJB Access Beans in the chain

Creating and Deploying Web Services using WSAD

Import the required Integration Objects and EJB Access Beans into Host Publisher Studio Application Integrator. Choose Create J2EE Archives to create the .ear file.

Follow the instructions in "Preparing to work with an Integration Object" on page 3, "Working with Host Publisher .ear, .war, and .jar files in WSAD" on page 4, and "Setting up the WebSphere Test Environment in WSAD" on page 5 (if you want to test your Web Service using the WebSphere Test Environment). If you complete these steps successfully, an application server instance exists where the Host Publisher Server runtime has started successfully. You also have folders that represent the following:

- The application_name.ear file you created with Host Publisher Studio Application Integrator
- The application_name.war file contained in the application_name.ear file
- If your .ear file contained EJB Access Beans, the EJB .jar file contained in the application_name.ear. Ensure that you generated the deployment code for the EJB .jar.

Creating Web Services from an Integration Object

1. Import the helper object (from Studio_Install_Dir\Studio\IntegrationObjects\WS\IOName\IONameHelperObjectSuffix.jar) into the WEB-INF\lib directory of your web application.
2. Update the Java Build path:
   - Add the XERCES classpath variable
3. Create a Web Service in your web application from the helper object:
   a. Create a Java bean Web Service from the *IONameHelperObjectSuffix*.jar.
   b. Use Request scope.
   c. Make processWSRequest the Web Service Java Bean method. Other methods are shown as choices, and you must deselect them.
   d. Choose to generate a sample to test your Web Service. When the sample is launched, choose the processWSRequest method.
   e. For inputs, set the following input properties in addition to the Integration Objects inputs you created in Host Access or Database Access:
      - hPubStartType=0
      - hPubEndType=0
      - hPubErrorOccurred=0
   f. After you have completed development and test of your application, export it from WSAD and deploy it to your Host Publisher Server.

Creating Web Services From an EJB Access Bean

1. Update the Java Build path:
   - Add the XERCES classpath variable
2. Create a Web Service in your Web application from the EJB Access Bean:
   a. Create a Java bean Web Service from the *IONameEJBAccessBeanSuffix*.jar.
   b. Use Request scope.
   c. Make processWSRequest the Web Service Java Bean method. Other methods are shown as choices, and you must deselect them.
   d. Choose to generate a sample to test your Web Service. When the sample is launched, choose the processWSRequest method.
   e. For inputs, set the following input properties in addition to the Integration Objects inputs you created in Host Access or Database Access:
      - hPubStartType=0
      - hPubEndType=0
      - hPubErrorOccurred=0
   f. After you have completed development and test of your application, export it from WSAD and deploy it to your Host Publisher Server.

Chapter 6. Using Remote Integration Objects

We recommend that you use the Web Services function as an alternative to Remote Integration Objects. Web Services represents IBM's strategic solution, providing an open, standards-based way to access Integration Objects from a remote machine. For more information about using Web Services, refer to the *IBM WebSphere Host Publisher Administrator's and User's Guide*.

Remote Integration Objects (RIOs) are designed for use by Java programmers familiar with Java code and using Host Publisher Integration Objects.

You can use Remote Integration Objects to access Integration Object data from a Java program (applet or application) running on a remote machine. The remote machine requires lightweight Remote Integration Object .jar files and network access to a Host Publisher Server; however, it does not require Host Publisher Server or WebSphere Application Server.

You can customize the lightweight Java program for specific business needs, such as correlating data with other Java beans or XML data sources. It can be distributed as a standalone Java application or to a Web server as a downloadable Java applet.

Creating Remote Integration Objects and the sample application

The following steps create a Java program (applet or application) to access Integration Object data on a remote machine. In these steps, *IOName* is the name you gave to the Integration Object, and *install_dir* is the directory in which Host Publisher is installed.

1. Start Host Access or Database Access and open the Integration Object you want to access remotely.
2. Ensure that **Create Remote Integration Object** is checked on the Host Access or Database Access Options menu.
3. Save the Integration Object using the Host Access or Database Access File menu. You can also select **Create Integration Object** from the Host Access File menu. Host Publisher creates the Remote Integration Object files. Refer to "Remote Integration Object files" on page 36 for the names and location of the files that are created.
4. The sample CustomApp*IOName*.class program will work as it is; however, you can edit the CustomApp*IOName*.java file, which is located in the *install_dir*\Studio\IntegrationObjects\RemoteIO*IOName*\ directory, to perform whatever task is required to access the Integration Object data.
5. If you edit the sample CustomApp*IOName*.java file, compile the file with the CompileCustomApp*IOName*.bat utility file.
6. Package the files needed to run the Remote Integration Object on the remote machine using the PkgRIO*IOName*.bat utility file, which creates a RIO*IOName*.zip file.
7. Copy the RIO*IOName*.zip file to the remote machine, using FTP or a similar program, and unzip the file. For a Java application, JDK™ 1.1.7 or higher is required for the remote machine to run the java application. For a Java applet, the RIO*IOName*.zip file should be transferred to the desired HTTP server for browser download.

© Copyright IBM Corp. 2000, 2002

8. Complete the Host Publisher application that uses the Integration Object in Host Publisher Studio, publish the application to Host Publisher Server, and deploy and start the application.

   Note: Host Publisher Studio does not require you to create JSPs to drive Integration Objects in your Host Publisher application. You might choose to import only Integration Objects into your application.

9. After unzipping the RIO*IOName*.zip file on the remote machine, you can run the CustomApp*IOName*.java file using the RunCustomApp*IOName*.bat utility file.

Programming with Remote Integration Objects

Host Publisher Integration Objects contain Java methods that you can use when programming with Integration Objects. In addition to the methods described in "Integration Object methods" on page 11, you can use the following methods when programming with Remote Integration Objects:

java.lang.String getURLString()
: Returns the URL used to access the Remote Integration Object servlet

void setURLString(java.lang.String urlString)
: Sets the URL used to access the Remote Integration Object servlet

java.lang.String getSessionID()
: Returns the session identifier that WebSphere assigns to the HTTP connection between the browser and the Web server from the first Integration Object in a chain. This method is analogous to the **java.lang.String getHPubLinkKey()** method described in "Integration Object methods" on page 11.

void setSessionID(java.lang.String newID)
: Sets the session identifier for the HTTP connection between the browser and the Web server for subsequent Integration Objects in a chain. This method is analogous to the **void setHPubLinkKey(java.lang.String value)** method described in "Integration Object methods" on page 11.

You can invoke a Remote Integration Object from a servlet like an Integration Object with the following exception:

- You must call RemoteTestIO.setURLString(String urlstring) with the URL pointing to RIOServlet on Host Publisher Server. You must invoke this prior to performing the processRequest() function. For example:

  ```
  RemoteTestIO.setURLString ("http://localhost/context_root/RIOServlet")
  ```

  where *context_root* is the context root of the application that contains the integration object.
- Remote Integration Objects support chained Integation Objects, but do not contain the doHPTransaction(HttpServletRequest, HttpServletResponse) method. Use the processRequest() method to perform the action, and use the getSessionID() and setSessionID() methods to get and set the session ID.

Using Remote Integration Objects

To use a Remote Integration Object, do the following:

1. Initialize and start the Host Publisher Server runtime, if it is not already running. You can start the Host Publisher Server runtime by defining it as a WebSphere Application Server custom service or by creating a servlet to start the runtime. See "Defining Host Publisher Server as a WebSphere custom

service" on page 10 for information on using custom services. See "Sample code for starting and stopping the Host Publisher Server runtime" on page 5 for sample code to include in a servlet.

2. Create an instance of the Remote Integration Object by calling its constructor. Set the URL to the location of RIOServlet on the target Host Publisher Server.

   ```
   RemoteTestIO.setURLString ("http://hpserver/context_root/RIOServlet")
   ```

   where *context_root* is the context root of the application that contains the integration object.

3. Invoke the methods for the Remote Integration Object instance. You might want to invoke methods to set properties of input variables. The naming convention for setter methods is as follows:

   ```
   RemoteTestIO.setXyz(String)
   ```

   where *xyz* is the name of your input variable.

4. Invoke the Remote Integration Object to perform its task (running a macro or querying a database, for example), using the method

   ```
   RemoteTestIO.processRequest()
   ```

5. Check for errors by invoking

   ```
   RemoteTestIO.getHPubErrorOccurred()
   ```

Remote Integration Object chaining

To chain Remote Integration Objects, do the following:

1. Create an instance of the first Remote Integration Object in the chain by calling its constructor. Set the URL to the location of RIOServlet on the target Host Publisher Server.

   ```
   RIOChain1.setURLString ("http://hpserver/context_root/RIOServlet")
   ```

   where *context_root* is the context root of the application that contains the integration object.

2. Invoke the methods for the Remote Integration Object instance. You might want to invoke methods to set properties of input variables. The naming convention for setter methods is as follows:

   ```
   RIOChain1.setXyz(String)
   ```

   where *xyz* is the name of your input variable.

3. Invoke the Remote Integration Object to perform its task (running a macro or querying a database, for example), using the method

   ```
   RIOChain1.processRequest()
   ```

4. Check for errors by invoking

   ```
   RIOChain1.getHPubErrorOccurred()
   ```

5. Extract and save the key that represents the connection for the Remote Integration Object chain

   ```
   String mySession = RIOChain1.getSessionID();
   ```

6. Create an instance of the next Remote Integration Object in the chain by calling its constructor. Set the URL to the location of RIOServlet on the target Host Publisher Server.

   ```
   RIOChain2.setURLString ("http://hpserver/context_root/RIOServlet")
   ```

   where *context_root* is the context root of the application that contains the integration object.

7. Set the key for this chained connection
   ```
   RIOChain2.setSessionID(mySession);
   ```
8. Invoke the methods for this Remote Integration Object instance. You might want to invoke methods to set properties of input variables. The naming convention for setter methods is as follows:
   ```
   RIOChain2.setXyz(String)
   ```
   where *xyz* is the name of your input variable.
9. Invoke this Remote Integration Object to perform its task (running a macro or querying a database, for example), using the method
   ```
   RIOChain2.processRequest()
   ```
10. Check for errors by invoking
    ```
    RIOChain2.getHPubErrorOccurred()
    ```

Repeat steps 6 through 10 for any and all subsequent Remote Integration Objects in the chain.

Obtaining Integration Object data in XML format

Refer to Figure 3 for an overview of the Remote Integration Object components and the interactions between them.

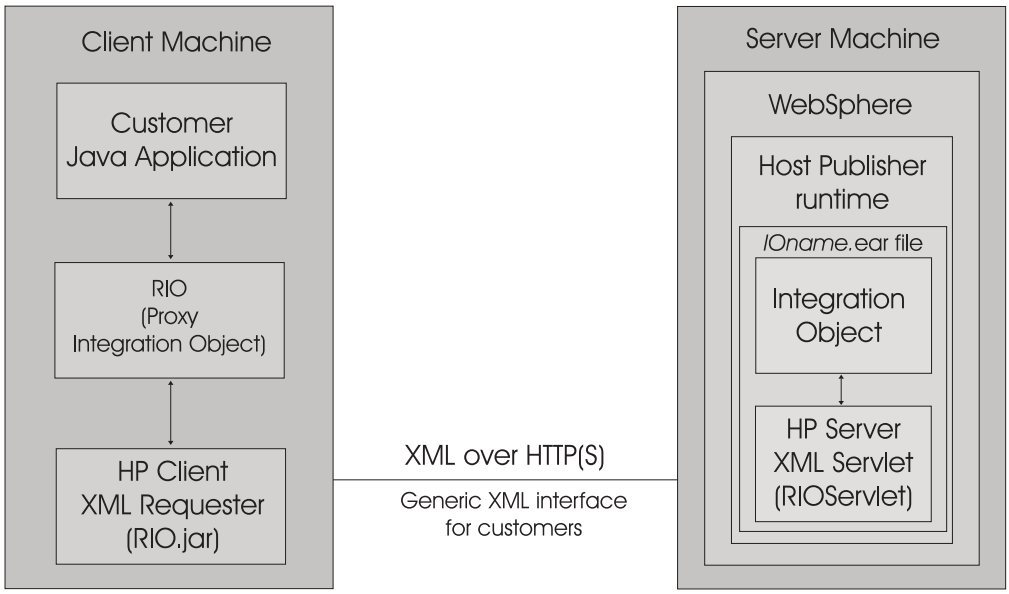


Figure 3. Remote Integration Object components

Integration Object data can be queried from an XML application and from a browser that supports XML data. The XML application requires an XML parser and TCP/IP connectivity to a Host Publisher Server. No other packaging is necessary. When **Create Remote Integration Object** on the Options menu in Host Access or Database Access is checked, a sample XML*IOName*.html file (where *IOName* is the name you gave to the Integration Object) is created that extracts Integration Object data in XML format. To extract the data in XML format, the XML browser or XML application must send a URL to the Host Publisher Web server as follows:

To request a list of input parameters:

```
http://yourserver/context_root/RIOServlet?hPubIntegrationObjectName=IntegrationObject.TestDB
       &hPubRequestType=requestInputs
```

where *context_root* is the context root of the application that contains the integration object.

To run an Integration Object with optional style sheet processing:

```
http://yourserver/context_root/RIOServlet?hPubIntegrationObjectName=IntegrationObject.TestDB
       &hPubRequestType=execute
       &hPubExecuteXML=&EXECUTEXMLDOC
       &hPubXMLServerStyleSheet=SERVERSTYLE
       &hPubXMLClientStyleSheet=CLIENTSTYLE
```

where *context_root* is the context root of the application that contains the integration object.

To run an Integration Object with input parameters and optional style sheet processing:

```
http://yourserver/context_root/RIOServlet?hPubIntegrationObjectName=IntegrationObject.TestDB
       &hPubRequestType=execute
       &INPUTNAME1=INPUTVAL1&INPUTNAME2=INPUTVAL2...
       &hPubXMLServerStyleSheet=SERVERSTYLE
       &hPubXMLClientStyleSheet=CLIENTSTYLE
```

where *context_root* is the context root of the application that contains the integration object.

Where: TestDB is the name of the Integration Object to run

SERVERSTYLE is the server style sheet to apply

CLIENTSTYLE is the client style sheet that the browser will apply

INPUTNAME1=INPUTVAL1... define the input parameters and values of the Integration Object

Notes:

1. Each new parameter in the URL begins with an ampersand (&). There should be no spaces in the URL. If you use either of the optional stylesheet parameters, do not type a space between the other parameters and the stylesheet parameters.
2. WebSphere Application Server provides two default style sheets: default.xsl and default2.xsl in the WebSphere directory.

   Host Publisher provides the HPubConvertToTableFormat.xsl style sheet in the *install_dir*\Common\ directory, where *install_dir* is the directory in which Host Publisher is installed. For more information, refer to "Chapter 2. Applying XML stylesheet processing to Integration Object output" on page 15.

   If you specify *SERVERSTYLE* in the URL, enter the full file path of the server style sheet, for example *\websphere\AppServer\web\xml\xsl\default\default.xsl*.
3. For *CLIENTSTYLE*, Host Publisher creates a sample style sheet (StyleSheet*IOName*.xsl where *IOName* is the name you give to the Integration Object) when you create a Remote Integration Object. Copy the StyleSheet*IOName*.xsl file from the *install_dir*\Studio\IntegrationObjects\RemoteIO*IOName*\ directory to a directory accessible through the URL. If you specify *CLIENTSTYLE* in the URL, enter the full file path of the location where you copied the StyleSheet*IOName*.xsl file.

The response from the Host Publisher Web server is the XML data defined by the following Data Type Declaration (DTD):

```
<?xml version="1.0" standalone="yes">
<!DOCTYPE com.ibm.HostPublisher.IntegrationObject.properties [
<!ELEMENT com.ibm.HostPublisher.IntegrationObject.properties
              (inputProperties, outputProperties)>
<!ATTLIST com.ibm.HostPublisher.IntegrationObject.properties name CDATA "">
<!ELEMENT inputProperties (inputProperty*)>
<!ELEMENT inputProperty (value)>
<!ATTLIST inputProperty name CDATA "">
<!ELEMENT outputProperties (outputProperty*)>
<!ELEMENT outputProperty (value*)>
<!ATTLIST outputProperty name CDATA "">
<!ELEMENT value (#PCDATA)>
]>
```

Remote Integration Object files

The Remote Integration Object file names are derived from the Integration Object file name. In the following example, the Remote Integration Object files were created in the `\HostPub\Studio\IntegrationObjects\RemoteIO\TestDB\` directory for an Integration Object named TestDB.

IntegrationObject\RemoteTestDB.java	Remote Integration Object proxy source file **Note:** This file should not be modified. If you want to change the function of the Remote Integration Object, modify the CustomApp*IOName*.java file.
IntegrationObject\RemoteTestDB.class	Remote Integration Object proxy class file
CustomAppTestDB.java	Sample Java program source that calls Remote Integration Object proxy class
CustomAppTestDB.class *	Compiled sample Java program
AppLoaderTestDB.html *	HTML file to load sample Java applet, CustomAppTestDB.class
XMLTestDB.html	HTML file to get Integration Object in XML format
StyleSheetTestDB.xsl	Sample client style sheet
CreateRIOJavaDoc.bat	Utility file that creates a Javadoc document for RemoteTestDB.java
CompileCustomAppTestDB.bat	Utility file that compiles CustomAppTestDB.java using JDK 1.2
PkgRIOTestDB.bat	Utility file that packages the Remote Integration Object files in a .zip file for transfer to a client machine
RunCustomAppTestDB.bat	Utility file that runs CustomAppTestDB.java using JDK 1.2
CreateRIOBeanJar.bat	Utility file that packages the IntegrationObject\RemoteTestDB.class file in a .jar file as a Java bean. This utility is useful for development tools that require .jar and META files for importing Java beans.
Note: * CustomAppTestDB.class and AppLoaderTestDB.HTML require access to some Swing files to run correctly: **JDK 1.1.7 or higher** classes.zip and swingall.jar **JDK 1.2 or higher** rt.jar and tools.jar	

Chapter 7. Customizing Host Access Integration Object Java code

Host Publisher enables you to modify how an Integration Object interacts with the underlying subsystems, at the Java code level, to perform additional functions. Host Publisher provides Java coding templates for building Host Access Integration Objects. You specify which coding templates to use in building an Integration Object by updating the HOD_BEAN_TEMPLATE_PATH and HOD_BEAN_INFO_TEMPLATE_PATH parameters in the Host Publisher Studio initialization file (Studio.ini). The HOD_BEAN_TEMPLATE_PATH is used to locate the template that defines the Java bean code used to create Integration Objects. The HOD_BEAN_INFO_TEMPLATE_PATH is used to locate the template that defines the Java BeanInfo file used to create Integration Objects. The Studio.ini file is stored in the *install_dir*\Studio directory, where *install_dir* is the directory in which Host Publisher Studio is installed.

There are two types of Java coding templates: default templates and customizable templates. The templates are also stored in the *install_dir*\Studio directory. The default templates are HPubTemplateHODBean.Default and HPubTemplateHODBeanInfo.Default. The customizable templates are HPubTemplateHODBean.Customize and HPubTemplateHODBeanInfo.Customize.

The Studio.ini file is automatically created when you access any of the three Host Publisher Studio components, Application Integrator, Database Access, and Host Access. The Studio.ini file specifies the default templates for creating Host Access Integration Objects. These templates contain Java code that is independent of the Host Publisher and Host On-Demand code. Integration Objects created using the default templates will not need to be recompiled and redeployed if the Host Publisher or Host On-Demand code changes for enhancements or service.

The customizable templates contain substantial Java code that interacts with the Host Publisher code, Host On-Demand objects, events, and other Java constructs. These templates enable you to modify an Integration Object to perform additional functions. Integration Objects created using the customizable templates will contain code that directly interacts with the Host Publisher and Host On-Demand code and implements much of the data processing. If any Host Publisher or Host On-Demand code changes affect the code contained in the Integration Object, the Integration Object will have to be recompiled and redeployed.

Using Java coding templates

If you do not need to modify how Integration Objects interact with Host Publisher or the operating environment, always use the HPubTemplateHODBean.Default and HPubTemplateHODBeanInfo.Default templates.

If you want all Host Access Integration Objects to perform additional functions, whether you are creating new Integration Objects or updating existing Integration Objects, use the HPubTemplateHODBean.Customize and HPubTemplateHODBeanInfo.Customize templates. Modify the templates to add Java code for the functions you want the Integration Objects to perform.

© Copyright IBM Corp. 2000, 2002

If you want only a small number of Integration Objects to perform additional functions, make a copy of the HPubTemplateHODBean.Customize and HPubTemplateHODBeanInfo.Customize templates and rename them. Modify the new template files to add Java code for the functions you want the Integration Objects to perform.

If you use either the customizable templates or renamed copies of the templates, update the Studio.ini file in the *install_dir*\Studio directory, where *install_dir* is the directory in which Host Publisher Studio is installed, to show the template names on the HOD_BEAN_TEMPLATE_PATH and HOD_BEAN_INFO_TEMPLATE_PATH path variables.

Modifying Java coding templates

The HPubTemplateHODBean.Customize and HPubTemplateHODBeanInfo.Customize templates contain Java code that is incorporated into the Integration Object Java bean code (.java) file when the Integration Object is compiled. The templates also contain constructs specifically for Host Publisher, prefaced with a percent sign (%). These constructs enable Host Publisher to create Java beans from the data specified by the Host Publisher Studio user when the Integration Object is created. When modifying the template files, be careful not to delete the statements containing the Host Publisher constructs. Make backup copies of the HPubTemplateHODBean.Customize and HPubTemplateHODBeanInfo.Customize templates before you begin making changes to the template files.

For example, suppose that you want to trace the name and the x and y screen coordinates of the Host On-Demand Extract Events that are processed by an Integration Object.

Note: Extraction of the x and y screen coordinates is not available in the Web Services, Remote Integration Objects, or EJB environments, because the x and y coordinates require access to internal variables not available in those environments.

Assuming that Host Publisher Studio was installed in C:\HostPub\, do the following:

1. Back up the file C:\HostPub\Studio\HPubTemplateHODBean.Customize
2. Change the code that extracts the macro event in C:\HostPub\Studio\HPubTemplateHODBean.Customize to add the following lines after the *pullVariableValueFromExtractData(haovWorkOnThis, data);...* statement:

```
// --- Trace X and Y screen coordinates example ---
if (HPubTracingOn) {
  String strg = "Extracting variable: " + stringExtractNameForThisEvent +
                " from screen location (" +
                haovWorkOnThis.intXScreenLocation + "," +
                haovWorkOnThis.intYScreenLocation + ")";
  Ras.trace( this.getClass().getName(),"macroExtractEvent", strg);
}
```

For example:

```
...
  public void macroExtractEvent(MacroExtractEvent oMacroExtractEvent)
  { // a HOD macroExtractEvent was fired for this macro

...
    pullVariableValueFromExtractData( haovWorkOnThis, data);
```

```
// --- Trace X and Y screen coordinates example ---
if (HPubTracingOn) {
  String strg = "Extracting variable: " + stringExtractNameForThisEvent +
                " from screen location (" +
                haovWorkOnThis.intXScreenLocation + "," +
                haovWorkOnThis.intYScreenLocation + ")";
  Ras.trace( this.getClass().getName(),"macroExtractEvent", strg);
}...
```

3. Update the path variables HOD_BEAN_TEMPLATE_PATH and HOD_BEAN_INFO_TEMPLATE_PATH in C:\HostPub\Studio\Studio.ini to:

   ```
   HOD_BEAN_TEMPLATE_PATH=C:\HostPub\Studio\HPubTemplateHODBean.Customize
   HOD_BEAN_INFO_TEMPLATE_PATH=C:\HostPub\Studio\HPubTemplateHODBeanInfo.Customize
   ```

4. Restart Host Publisher Studio.
5. Create an Integration Object as you normally would. If you want to modify an existing Integration Object to trace the name and the x and y screen coordinates of the Host On-Demand Extract Events, open the existing Integration Object and re-create it by using the **Create Integration Object** selection on the **File** menu in Host Access.

Debugging customizable Host Access Integration Object compilation errors

If the new code that was added to a customizable template causes a compilation error, you will receive a Host Publisher Studio message stating that the object could not be created. A file named `iofailed.txt` in the *`install_dir`*`\Studio\` directory will contain a copy of the compilation errors.

A common class for accessing Host Access Integration Object information

The properties of an Integration Object can be accessed from WebSphere applications. The calling program must know the name of the Integration Object and the name of the variable. Sometimes, it is advantageous for the calling program to be able to access properties that all Integration Objects share, without knowing the name of the Integration Object. A new Java class, HPubHostAccess, has been added and is extended by all Host Access Integration Objects. The HPubHostAccess class contains properties common to Host Access Integration Objects. The programmer can extract information from the HPubHostAccess class without knowing the name of the Integration Object. The HPubHostAccess class can be introspected to find the name of the current properties that can be extracted using the Integration Object methods. For information on these methods, see “Integration Object methods” on page 11.

Java class hierarchy of Host Access Integration Objects

Following is the Java class hierarchy of the default and customizable Integration Objects:

```
HPubCommon --  HPubHostAccess + -- HPubHODCommon -- HPubTemplateHODBean.Default
                               |
                               + -- HPubTemplateHODBean.Customize
```

Chapter 8. Customizing JavaServer Page (JSP) migration

With Host Publisher Version 2, JSPs were created using the JSP 0.91 specification. With Host Publisher Version 3.5, JSPs were created using the JSP 1.0 specification. Host Publisher Version 4.0 creates JSPs using the JSP 1.1 specification. You must migrate JSPs that were created using the JSP 0.91 specification. You do not need to migrate JSPs that use JSP 1.0 or 1.1 specifications.

Host Publisher Version 4.0 provides a JSP migration utility, JSPMigrator utility, that is invoked when the AppMigrator and StudioAppMigrator utilities run. The JSP migration utility converts all 0.91 JSP tags generated by Host Publisher Studio or the Host Publisher error page to JSP 1.1 tags or to inline Java code. Following is a list of 0.91 JSP tags and how they are migrated.

<BEAN>

Replaced with the JSP 1.1 <jsp:useBean> tag

```
<jsp:useBean        id="dBAcc"        type="IntegrationObject.DBAcc"
class="IntegrationObject.DBAcc" scope="request"> </jsp:useBean>
```

<INSERT></INSERT>

Information between <INSERT> and </INSERT> is replaced with in-line Java code. For example:

```
<%= dBAcc.getDB2ADMINEMPLOYEEBIRTHDATE_(_i0) %>
```

<REPEAT></REPEAT>

Information between <REPEAT> and </REPEAT> is replaced with in-line Java code.

<REPEAT> is replaced with:

```
<%
for (int _i0 = 0; _i0 <= 2147483647;_i0++){
try {
%>
```

where *_i0* is the name of the index used in the original REPEAT tag.

</REPEAT> is replaced with:

```
<%
}
catch (java.lang.ArrayIndexOutOfBoundsException _e0)
 {
  break;
 }
catch (Java.lang.NullPointerException _e)
 {
  break;
 }
}
%>
```

<%@ content_type="text/html;charset=ISO-8859–1" %>

Replaced with the following JSP 1.1 syntax:

```
<%@ page contentType="text/html;charset=ISO-8859–1" %>
```

In Host Publisher-created error pages:

- The word **session** is converted to **hp_session**.

© Copyright IBM Corp. 2000, 2002

- The tag **com_ibm_HostPublisher emsg** is converted to **com_ibm_HostPub emsg**.

If you have added JSP 0.91 tags to Web pages in your applications other than those described here, those tags are not migrated to JSP 1.1 tags. The log file for the JSPMigrator utility identifies tags and code that are not converted. You must determine if these tags need to be migrated manually.

Host Publisher includes sample source code for the migration utility in the *install_dir*\SDK\JSPCustomMigration\ directory, where *install_dir* is the directory in which Host Publisher is installed. You can modify the code to include migration of JSP tags that you have added to your Web pages. HTML instructions for modifying the JSPCustomMigrator.java file are also included in the *install_dir*\SDK\JSPCustomMigration\ directory. Point your Web browser to *install_dir*\SDK\JSPCustomMigration\Introduction.html to view the instructions.

Note: If you modify the JSPCustomMigrator code, your modified version of the code is not used during installation of Host Publisher Server nor for JSP migration in Host Publisher Studio. You must run your modified version of the code from the command line. In addition, the modified version of the code must be run before the AppMigrator and StudioAppMigrator utilities are initiated. If not, when the AppMigrator and StudioAppMigrator utilities run, errors in JSP pages are reported.

Chapter 9. Host Publisher File formats

When you use the Transfer to Server function in the Host Publisher Application Integrator, the application's Web pages, Java objects, and other resources are assembled into a binary J2EE-compliant Enterprise Archive (.ear) file, which contains a J2EE-compliant Web Application Record (.war) file. If you requested EJB support when you created the Integration Object in Host Publisher Studio, the .ear file also contains and EJB .jar file.

By default, the .ear file is stored in the *Studio_Install_Dir*\Studio\Applications*app_name* subdirectory path, where *app_name* is the name of your application. The .ear file is transferred to a Host Publisher Server, to the *WebSphere_Install_Dir*/installableApps/HostPublisher directory. After the administrator deploys and starts the application's .ear file, WebSphere Application Server uses the files to service end-user requests to Host Publisher.

The J2EE specification provides details about the contents and layout of .ear files, .war files, and EJB .jar files.

Host Publisher produces applications using standard open formats—such as HTML pages, Java files, JSPs, and XML files. This makes it simple to make changes to a Host Publisher application after it has been published to a Host Publisher Server. You don't have to keep returning to Host Publisher Studio to make small changes to your application.

Warning: If you make changes to applications on the server without updating the files in Host Publisher Studio, you could lose the changes in the server version when you next publish your application. Be sure to update the version you keep in Host Publisher Studio before you publish those files. There is no automatic way to synchronize the two versions.

A Host Publisher application is made up of several types of files. The following sections describe each file, explain how it is used, and provide the file format. Some of these files are stored in the .war file, or the EJB .jar file for EJB-based applications, which are contained in the .ear file, when the application is transferred to a server. Other files are specific to Host Publisher Studio and are not transferred to the server.

Integration Object project (.hpi) file

Integration Object project (.hpi) files are specific to Host Publisher Studio and are not transferred to the server.

Host Access and Database Access store project information into .hpi files. These files describe the details you defined while creating an Integration Object.

Note: The format of the Integration Object project (.hpi) file is shown for information only. If you manually edit this file, you might receive unexpected results.

The following is a sample .hpi project file generated by Host Access.

© Copyright IBM Corp. 2000, 2002

```
<?xml version="1.0" standalone="yes"?>

<com.ibm.HostPublisher.IntegrationObject name = "Pat1"   type="hod">

  <Package name = "IntegrationObject"/>
  <Session>
    <PoolName>callup</PoolName>
    <FileName>Callup.poolspec</FileName>
  </Session>

  <EJB PropertiesSuffix = "Properties" HelperSuffix = "Helper"
              EJB10AccessBeanSuffix = "Access0"
              EJB11AccessBeanSuffix = "Access1"/>

  <RIO RIOPrefix = "Remote"/>
  <OutputVariable name="table1" type ="simple">
    <ScreenCoordinates x="1" y="14" dx="79" dy="1"/>
      <SubVariable name="column0" type ="array">
        <RelativeCoordinates x="0" y="0" dx="7" dy="1"/>
      </SubVariable>
      <SubVariable name="column1" type ="array">
        <RelativeCoordinates x="7" y="0" dx="5" dy="1"/>
      </SubVariable>
      <SubVariable name="column2" type ="array">
        <RelativeCoordinates x="12" y="0" dx="12" dy="1"/>
      </SubVariable>
      <SubVariable name="column3" type ="array">
        <RelativeCoordinates x="24" y="0" dx="9" dy="1"/>
      </SubVariable>
      <SubVariable name="column4" type ="array">
        <RelativeCoordinates x="33" y="0" dx="4" dy="1"/>
      <SubVariable name="column5" type ="array">
        <RelativeCoordinates x="37" y="0" dx="9" dy="1"/>
      <SubVariable name="column6" type ="array">
        <RelativeCoordinates x="46" y="0" dx="9" dy="1"/>
      <SubVariable name="column7" type ="array">
        <RelativeCoordinates x="55" y="0" dx="18" dy="1"/>
      <SubVariable name="column8" type ="array">
        <RelativeCoordinates x="73" y="0" dx="6" dy="1"/>
      </SubVariable>

  </OutputVariable>

  <HODMacro filename = "callup.macro"/>
  <SessionChain>
    <StartState name = "Start Label"/>
    <EndState name = "The End Label"/>
    <Position>middle</Position>
  </SessionChain>
<com.ibm.HostPublisher.IntegrationObject>
```

The following is a sample .hpi project file generated by Database Access.

```
<?xml version="1.0" standalone="yes"?>

<com.ibm.HostPublisher.IntegrationObject name = "QuerySample"   type="db">

  <Package name = "IntegrationObject"/>

  <Session>
    <FileName>SessionDefs.XML</FileName>
    <PoolName>QuerySample</PoolName>
  </Session>

  <EJB PropertiesSuffix = "Properties" HelperSuffix = "Helper"
              EJB10AccessBeanSuffix = "Access0"
              EJB11AccessBeanSuffix = "Access1"/>
```

```
  <RIO RIOPrefix = "Remote"/>

  <SQL>
        SELECT "JMYERS"."DEPARTMENT"."DEPTNO", "JMYERS"."DEPARTMENT"."DEPTNAME",
               "JMYERS"."DEPARTMENT"."MGRNO" FROM "JMYERS"."DEPARTMENT"
                WHERE ( ( "JMYERS"."DEPARTMENT"."DEPTNO" = 'B01' ) AND
                ( "JMYERS"."DEPARTMENT"."DEPTNAME" = @+)deptname@-)''@+) ) )
  </SQL>

  <JDBCUrl name="jdbc:db2:sample"/>
  <JDBCDriver name="COM.ibm.db2.jdbc.app.DB2Driver"/>
<com.ibm.HostPublisher.IntegrationObject>
```

Tag descriptions:

com.ibm.HostPublisher.IntegrationObject

name Specifies the name of this Integration Object. This name must match the name of the file.

type The type of Integration Object described in this file. Valid values are **hod** or **db**.

EJB Specifies the suffixes appended to the name of the Integration Object for naming EJB Access Beans and object files. You specify the values for the attributes using the EJB Integration Object Properties selection of the Options menu in Host Publisher Studio when creating the Integration Object. The attributes are:

EJB10AccessBeanSuffix
Specifies the suffix appended to the name of the Integration Object for naming the EJB 1.0 Access Bean files. The default value is Access0, unless you modify the value in the Studio.ini file.

EJB11AccessBeanSuffix
Specifies the suffix appended to the name of the Integration Object for naming the EJB 1.1 Access Bean files. The default value is Access1, unless you modify the value in the Studio.ini file.

HelperSuffix
Specifies the suffix appended to the name of the Integration Object for naming the EJB helper object files. The default value is Helper, unless you modify the value in the Studio.ini file.

PropertiesSuffix
Specifies the suffix appended to the name of the Integration Object for naming the EJB properties object files. The default value is Properties, unless you modify the value in the Studio.ini file.

HODMacro

filename
Identifies the filename containing the Host On-Demand macro recorded by the user for this Host Access Integration Object.

JDBCDriver

name The JDBC driver name to use for the specified URL for Database Access Integration Objects.

JDBCUrl

name The JDBC URL to connect to when executing this Database Access Integration Object.

OutputVariable

Output variables define data that will be extracted during the macro execution in Host Access Integration Objects.

name Specifies the variable name provided by the user during macro recording.

type Specifies the type of variable described by the tag. Valid values are **simple** and **array**. Simple variables are stored and displayed as single blocks of text. Array variables are stored as individual lines that can be displayed individually on a JSP.

ScreenCoordinates

Identifies a rectangular region on the application screen that defines the data to extract.

x Identifies the starting column number. The first column begins at 1.

y Identifies the starting row number. The first row begins at 1.

dx Identifies the total number of columns to include in this variable.

dy Identifies the total number of rows to include in this variable.

Package

name Specifies the Java package name used when generating the Java source code for this object.

RIO

RIOPrefix

Specifies the prefix prepended to the name of the Integration Object for naming the Remote Integration Object files. You specify the value for the RIOPrefix attribute using the Remote Integration Object Properties selection of the Options menu in Host Publisher Studio when creating the Integration Object. The default value is Remote, unless you modify the value in the Studio.ini file.

Session

PoolName

Specifies the connection pool name used by this Integration Object.

FileName

Specifies the file name where the pool name is stored.

SessionChain

StartState name
The connection start state label given by the user for the Host Access Integration Object.

EndState name
The connection end state label given by the user for the Host Access Integration Object.

Position
The position of this Host Access Integration Object in the object chain. Valid values are **first**, **middle**, or **last**.

SQL Identifies the SQL statement to execute for Database Access Integration Objects.

SubVariable Subvariables define further detail of the format of data defined by an output variable. Subvariables are generated when the user specifies data to be extracted as a table in Host Access. Each column identified by the user is defined using the SubVariable tag.

name Specifies the variable name provided by the user during macro recording.

type Specifies the type of variable described by the tag. Valid values are **simple** and **array**. Simple variables are stored and displayed as single blocks of text. Array variables are stored as individual lines that can be displayed individually on a JSP.

RelativeCoordinates
Identifies a rectangular region on the application screen that defines the data to extract.

x Identifies the starting column number. The first column begins at 1.

y Identifies the starting row number. The first row begins at 1.

dx Identifies the total number of columns to include in this variable.

dy Identifies the total number of rows to include in this variable.

Host Publisher application (.hpa) file

Host Publisher application (.hpa) files are specific to Host Publisher Studio and are not transferred to the server.

This XML file organizes all of the parts that make up a Host Publisher application, including Java objects and the Web pages that refer to them. Host Publisher applications are published to Host Publisher Servers for access by your customers. When Host Publisher Studio is used to load an existing application, it is this file that you actually open.

Note: The format of the Host Publisher application (.hpa) file is shown for information only. If you manually edit this file, you might receive unexpected results.

Here is a sample of a typical application file:

```
<?xml version='1.0' encoding='UTF-8'?>
<application>
<appl_name>EmployeeQuery</appl_name>
<integration_object>
  <obj_name>HostPub41\Studio\IntegrationObjects\EmployeeQuery.jar
  </obj_name>
  <input_properties>
    <input>setEmpno</input>
  </input_properties>
  <output_properties>
    <output>getJMYERSEMPLOYEEEMPNO_</output>
    <output>getJMYERSEMPLOYEEFIRSTNME_</output>
    <output>getJMYERSEMPLOYEEMIDINIT_</output>
    <output>getJMYERSEMPLOYEELASTNAME_</output>
    <output>getJMYERSEMPLOYEEWORKDEPT_</output>
    <output>getJMYERSEMPLOYEEPHONENO_</output>
    <output>getJMYERSEMPLOYEEHIREDATE_</output>
    <output>getJMYERSEMPLOYEEJOB_</output>
    <output>getJMYERSEMPLOYEEEDLEVEL_</output>
    <output>getJMYERSEMPLOYEESEX_</output>
    <output>getJMYERSEMPLOYEEBIRTHDATE_</output>
    <output>getJMYERSEMPLOYEESALARY_</output>
    <output>getJMYERSEMPLOYEEBONUS_</output>
    <output>getJMYERSEMPLOYEECOMM_</output>
  </output_properties>
  <execution_method>doHPTransaction</execution_method>
</integration_object>
<page>D:\HostPub41\Studio\Applications\EmployeeQuery\output.jsp</page>
<page>D:\HostPub41\Studio\Applications\EmployeeQuery\input.jsp</page>
<connection_pool>D:\HostPub41\Studio\SessionDefs\Empno.poolspec</connection_pool>
</application>
```

Tag descriptions:

appl_name
: Names the Host Publisher application. This name must match the name of the file. It is also the name Host Publisher Server uses to track this application.

connection_pool
: Identifies an additional connection pool packaged with this application.

execution_method
: Specifies the Java method for invoking the Java object once the inputs are satisfied with data. After the execution method completes, the Java object's resulting data can be accessed using its output methods, if there are any.

input
: Specifies the Java method used to set an input value. For a Java bean, this is typically the setter method for a Java bean property.

input_properties
: Specifies the beginning of the list of inputs for this Java object. Inputs generally must be satisfied with data before the Java object can be executed. Each input is specified by a separate input tag under this tag.

integration_object
: Specifies beginning of a definition of an Integration Object or other Java object that was imported into Host Publisher Studio.

obj_name
: Specifies the full path to the Integration Object or other Java object within the file system. If the object is an Integration Object created using one of the Host Publisher Access applications, this file refers to a .jar file

containing the Integration Object Java bean and its related files. If this object is another Java object, this file refers to the file containing that Java object.

output
Specifies the Java method used to get data values from a Java object. For a Java bean, this is typically the getter method for a Java bean property.

output_properties
Specifies the beginning of the list of outputs for this Java object. Outputs are used to render Java object data within a Web page. Each output is specified by a separate output tag under this tag.

page Specifies a Web page that is used, either directly or indirectly, to access Java objects.

Integration Object source (.java) file

Integration Object source (.java) files are specific to Host Publisher Studio and are not transferred to the server.

Integration Objects created by Host Publisher Studio are Java beans. The Java bean files are contained within a .jar file and are generally made up of two files, the Java bean class and the Java bean BeanInfo class. These class files are generated based on a template maintained by Host Publisher Studio and information provided by you through one of the Host Publisher Access applications.

The template for Database Access Integration Obects is not available to you for customization, and since any time the Integration Object is modified using the Database Access application it is regenerated and compiled, do not customize the source files for the Integration Objects in any way. Instead, if you require custom logic to make use of Integration Object data, use JSP tags and additional Java code to include the logic in the Web pages, or develop another Java class that extends your Integration Object to customize Integration Object results.

Integration Objects created by Host Publisher Studio are Java beans. The Java bean files are contained within a .jar file and are generally made up of two files, the Java bean class and the Java bean BeanInfo class. These class files are generated based on templates maintained by Host Publisher Studio and information you provide through one of the Host Publisher Access applications.

There are two types of templates for Host Access Integration Objects: default templates and customizable templates. Host Access Integration Objects use the default templates by default. Default templates are not customizable. If you want to modify the way an Integration Object is generated, you must use a customizable template. The two customizable templates are HPubTemplateHODBean.Customize and HPubTemplateHODBeanInfo.Customize. For more information, see “Chapter 7. Customizing Host Access Integration Object Java code” on page 37.

The templates for Database Access Integration Obects are not customizable. Using the templates, the Integration Objects are regenerated and compiled every time the Integration Objects are modified using the Database Access application. If you require custom logic to make use of Database Access Integration Object data, use JSP tags and additional Java code to include the logic in the Web pages, or develop another Java class that extends your Integration Object to customize Integration Object results.

JavaServer Pages (JSP) Web page files

JSP Web page files are stored in the .ear file and are transferred to the server.

Host Publisher Studio generates JSP 1.1 pages to manipulate Java objects and their output. JSP tags are similar to HTML tags, but their purpose is to instantiate Java objects, execute methods, and access the object's properties (inputs and outputs). JSP tags enable you to interact with Java objects using standard Web pages.

With Host Publisher Version 2, JSPs were created using the JSP 0.91 specification. Host Publisher Version 4.0 creates JSPs using the JSP 1.1 specification. Host Publisher provides a migration utility to convert .91 JSPs to 1.1 JSPs. This utility can be invoked at server installation time, in Host Publisher Studio, and as a command line utility. See the *IBM WebSphere Host Publisher Administrator's and User's Guide* for more information about the migration utility.

If you want to customize JSP migration, Host Publisher provides sample JSP migration code. For more information about customizing the sample JSP migration code, see "Chapter 8. Customizing JavaServer Page (JSP) migration" on page 41.

The following are sample JSPs, followed by a description of how the tags are being used. If you edit any of the JSP Web page files in a non-English environment, you must use a UTF-8 capable editor.

JSP for the EmployeeQuery Integration Object

```
<HTML>
<%@ page contentType="text/html;charset=UTF-8" errorPage="DefaultErrorPage.jsp" %>
<BODY>

<jsp:useBean id="EmployeeQuery" type="IntegrationObject.EmployeeQuery"
                class="IntegrationObject.EmployeeQuery" scope="request">
<jsp:setProperty name="EmployeeQuery" property="*" />
</jsp:useBean>
<% EmployeeQuery.setHPubStartPoolName("EmployeeQuery"); %>
<% EmployeeQuery.doHPTransaction(request, response); %>

<P>Employee Data
<TABLE BORDER>
<TBODY>
<tr>
<th>Employee Number<th>
<th>First Name<th>
<th>Last Name<th>
<th>Phone Number<th>
</tr>
<%
for (int idx1 = 0; idx1 <= 2147483647; idx1 ++){
    try {
        String str =
            "<tr>" +"\n" +
            "<td>" + EmployeeQuery.getJMYERSEMPLOYEEEMPNO_(idx1) + "\n" +
            "<td>" + EmployeeQuery.getJMYERSEMPLOYEEFIRSTNME_(idx1) + "\n" +
            "<td>" + EmployeeQuery.getJMYERSEMPLOYEELASTNAME_(idx1) + "\n" +
            "<td>" + EmployeeQuery.getJMYERSEMPLOYEEPHONENO_(idx1) + "\n" +
            "<tr>" + "\n";
%>
<%= str %>
<%
       }
       catch (java.lang.ArrayIndexOutOfBoundsException e){
           break;
       }
```

```
       catch (java.lang.NullPointerException e){
           break;
       }
}
%>
</TBODY>
</TABLE>
</BODY>
</HTML>
```

JSP for the QuerySample Integration Object

```
<%
//-----------------------------------------------------------------------------
HttpSession hp_session =  request.getSession(true);
//-----------------------------------------------------------------------------
%>
<HTML>
<%@ page contentType="text/html;charset=UTF-8" errorPage="DefaultErrorPage.jsp" %>
<BODY>

<jsp:useBean id="QuerySample" type="IntegrationObject.QuerySample"
                class="IntegrationObject.QuerySample" scope="request">
<jsp:setProperty name="QuerySample" property="*" />
</jsp:useBean>
<% QuerySample.setHPubStartPoolName("QuerySample"); %>
<% QuerySample.doHPTransaction(request, response); %>

<FORM NAME="input" METHOD="POST" ACTION="<%= response.encodeUrl("input.jsp") %>">
<LABEL>Department</LABEL>
<SELECT NAME ="JMYERSDEPARTMENTDEPTNO_" MULTIPLE SIZE=3>
<%
for (int idx1 = 0 ; idx1 <= 2147483647; idx1 ++){
    try{
        String str =
             "<OPTION VALUE=\""
             + QuerySample.getJMYERSDEPARTMENTDEPTNO_(idx1) + "\"> "
             + QuerySample.getJMYERSDEPARTMENTDEPTNO_(idx1) + "\n";
%>
<%= str %>
<%
      }
      catch (java.lang.ArrayIndexOutOfBoundsException e){
         break;
      }
      catch (java.lang.NullPointerException e){
         break;
      }
}
%>
</SELECT>
<BR>
<INPUT TYPE="submit" VALUE="Submit">
</FORM>
</BODY>
</HTML>
```

The first page references an Integration Object called EmployeeQuery. After invoking the object, it renders the object's output in an HTML table with three columns. The second page references an Integration Object called QuerySample. After invoking the object, it renders the object's output in an HTML form that enables the user to select a department. The following tags are used on these pages:

FORM

The FORM tag encompasses the content of an HTML *fill-in form*. Use this

tag to create fill-in forms with checkboxes, radio buttons, and text input windows. It contains the following parameters:

ACTION
The ACTION parameter specifies the URL to which the FORM tag content is sent. This parameter is required.

METHOD
When the ACTION parameter indicates an HTTP URL, the METHOD parameter identifies the HTTP method for sending information to the server. This parameter is optional. Values for this parameter are:

GET The form content is appended to the URL.

POST The form content is sent to the server as a message body, and not as part of the URL.

NAME
The NAME parameter specifies the URL to which the FORM tag content is sent. This parameter is required.

Inline Java tag (<% %>)
Inline Java tags specify the beginning and end of Java code segments that are to be invoked as they are written. These segments may reference variables specified within other inline Java tags before these on the same page. As shown in the examples, these tags can be used to access or execute Java objects explicitly.

INPUT
The INPUT tag specifies a variety of editable fields inside a form. It contains the following parameters:

NAME
The NAME parameter specifies the variable name for the VALUE parameter.

TYPE The TYPE parameter specifies the type for the INPUT tag. This parameter is required. Values for this parameter are:

checkbox
INPUT tag elements are boolean quantities. The default value is off.

file The INPUT tag element is a file selection tool, with which the user can select a file to be sent with the FORM tag.

hidden
The INPUT tag element is not displayed to the user.

image The INPUT tag element is an active inline image.

password
The INPUT tag element is a single-line text field, but the text typed in the field is obscured by asterists or some other method. This is used for password entry.

radio The INPUT tag element is a radio button. Radio buttons are linked together by the same NAME parameter.

reset The INPUT tag element is a reset button. When pressed, all the fields in the FORM tag are reset to the values given by their VALUE parameter, erasing all user input.

submit

The INPUT tag element is a Submit button. Pressing the Submit button sends the FORM tag data to the specified URL.

text The INPUT tag element is a single-line text entry field. The physically displayed size of the input field is set by the SIZE attribute.

VALUE

The VALUE parameter specifies the initial value of the INPUT tag.

jsp:setProperty

The jsp:setProperty tag sets the value of one or more properties in a Java bean component, using the Java bean's *set* methods. You must use a jsp:useBean tag to declare the Java bean before you use the jsp:setProperty tag. The jsp:setProperty tag contains the following parameters:

name The name parameter names an instance of a Java bean that has already been created or located with a jsp:useBean tag. The value of the name parameter must match the value of the id parameter on a jsp:useBean tag. The jsp:useBean tag must appear before the jsp:setProperty tag in the same JSP file.

property

The property parameter sets the property values in a Java bean component. You can set property values in several ways:

- By passing all of the values in the user's request (stored as parameters in the *request* object) to matching properties in the Java bean
- By passing a specific value in the *request* object to a matching property or a property of a different name in the Java bean
- By explicitly setting a Java bean property to a value specified as a *String* or the result of an expression.

Each method of setting property values is determined by the values you specify on the property tag. The values are:

property="*"

Stores all of the values in the *request* object parameters (called request parameters) in matching Java bean properties. The property names in the Java bean must match the request parameters. The parameter names usually come from the elements of an HTML form, and the values come from the data the user enters.

The values of the request parameters are always of type *String*. The *String* values are converted to other data types so they can be stored in Java bean properties. The allowed Java bean property types and their conversion methods are shown in the following table:

Table 2. String conversions

Property Type	String Is Converted Using
boolean or Boolean	java.lang.Boolean.valueOf(String)
byte or Byte	java.lang.Byte.valueOf(String)
char or Character	java.lang.Character.valueOf(String)
double or Double	java.lang.Double.valueOf(String)

Table 2. String conversions (continued)

Property Type	String Is Converted Using
integer or Integer	java.lang.Integer.valueOf(String)
float or Float	java.lang.Float.valueOf(String)
long or Long	java.lang.Long.valueOf(String)

You can also use jsp:setProperty tag to set the value of an indexed property in a Java bean. The indexed property must have one of the types shown in Table 2 on page 53, and the request value assigned to it must be an array of the same type. The array elements are converted using the conversion methods shown in Table 2 on page 53.

If a request parameter has an empty or null value, the corresponding Java bean property is not set. If the Java bean has a property that does not have a matching request parameter, the property value is not set.

property=*"propertyName"* **[param=***"parameterName"* **]**
Sets one Java bean property to the value of one request parameter. The request parameter can have a different name than the Java bean property, and if so, you must specify the param attribute. If the Java bean property and request parameter have the same name, you can omit the param attribute.

If the parameter has an empty or null value, the corresponding Java bean property is not set.

You cannot use both the param and value attributes in a jsp:setProperty tag.

property=*"propertyName"* **value="{** *string* **\| <%=** *expression* **%> }"**
Sets one Java bean property to a specific value. The value can be a *string* or an *expression*. If you use a *string*, it is converted to the Java bean property's data type, according to the conversion rules shown in Table 2 on page 53. If you use an *expression*, the data type of the value of the expression must match the data type of the Java bean property.

If the parameter has an empty or null value, the corresponding Java bean property is not set.

You cannot use both the param and value attributes in a jsp:setProperty tag.

jsp:useBean
The jsp:useBean tag locates or instantiates a Java bean with a specific name and scope. The body of a jsp:useBean tag often contains a jsp:setProperty tag that defines property values in the object.

The jsp:useBean tag works with JavaBeans™ components, but not with enterprise beans. If you want to use enterprise beans, you can write a JSP file that constructs a JavaBean component, and have the JavaBean component call the EJB.

Note: In Host Publisher, Integration Objects and EJB Access Beans are both Java beans.

The jsp:useBean tag contains the following parameters:

class=*package.class*
The class parameter instantiates a Java bean from a class, using the *new* keyword and the class constructor. The *class* must not be abstract and must have a public, no-argument constructor. The *package* and *class* names are case sensitive.

id The id parameter names a variable that identifies the Java bean in the scope you specify. You can use the variable name in expressions or scriptlets in the same JSP file. The name is case sensitive and must conform to the naming conventions of the page scripting language.

scope The scope parameter defines a scope in which the Java bean exists and the variable named on the id parameter is available. Values for this parameter are:

application
The Java bean is set as a context in the application by a servlet that invokes the JSP file. If the Java bean is not part of the request context, the Java bean is created and stored in the request context.

page If the Java bean is present in the current JSP, the Java bean is reused. If the Java bean is not present, it is created and stored until the request in the current page is completed.

request
The life of the Java object lasts as long as the request for this page is being processed. It is discarded when a new page is requested.

session
The instance of this class is maintained past the current request, allowing other JSPs, for example, to access this object again.

The default value for the scope parameter is page.

type=*package.class*
If the Java bean already exists in the specified scope, the type parameter gives the Java bean the type you specify. If you use the type parameter without the class parameter, no Java bean is instantiated. The *package* and *class* names are case sensitive.

OPTION
The OPTION tag sets the different character-string options for a SELECT tag. The OPTION tag can contain characters, character references, or entity references. The VALUE attribute specifies the value assigned to the OPTION tag.

SELECT
The SELECT tag enables the user to select from a set of values presented as a selectable list of text strings, specified by the OPTION tag. The SELECT tag contains the following parameters:

MULTIPLE
The MULTIPLE parameter specifies that the user can select multiple items from a single SELECT tag. If MULTIPLE is not specified, the user can select only a single item from the SELECT tag. This parameter is optional.

NAME
This parameter specifies the variable name associated with the SELECT tag. This parameter is required.

SIZE This parameter specifies the number of displayed text lines. The default is 1, and the list is often presented as a pull-down menu.

Connection and configuration files

Connection and configuration files are stored in the .ear file and are transferred to the server.

This section describes the format of configuration files used by Host Publisher Server Administration to configure Host Publisher. The files use XML tags to structure their content. Host Publisher generates these files along with Integration Objects and publishes them to the server as part of an application. The configuration files are the following:

Connection specification (.connspec)
This file specifies the parameters necessary for establishing a connection to a data source, such as a 3270 application or a database.

Connection pool specification (.poolspec)
This file defines how to create a pool of connections to a host or database. It specifies parameters for pools of connections to data sources, such as 3270 applications or databases. It also serves as the main coordinating file for a complete connection pool definition (including connection, users, and connect and disconnect macros, if appropriate).

Logon specification (.logonspec)
This file specifies the names of the connect and disconnect macros for Host Access Integration Objects. If connection pooling is enabled, this file also specifies the name of the checkin screen.

User pool specification (.userpool)
This file lists the users and any associated user-specific information necessary for accessing a data source. It is this list of users and the connection definition that define a pool of connections.

Checkin screen description (.screen)
This Host On-Demand screen description identifies the host screen that should be active for a connection to be considered ready to be returned to the connection pool. If a connection is not in that state, it is discarded or recycled in an attempt to return the connection to that state. If connection pooling is not enabled, the checkin screen is ignored.

Macro files (.macro)
For Host Access Integration Objects only, these files specify IBM Host On-Demand keyboard and screen recognition macros. They are used for replaying sequences of keystrokes for performing certain tasks for the Integration Object, such as logging on to a system or accessing a data screen on an application. See “Macro script syntax” on page 66 for more information on the format of these XML files.

Notes:

1. Configuration descriptions might refer to other files that can be referenced using relative path names. The forward slash (/) is used as a file name separator, and it is replaced by the platform-specific filename separator character when Host Publisher Server Administration processes file names.
2. If you edit any of the connection and configuration files in a non-English environment, you must use a UTF-8 capable editor.
3. Because applications do not share connection and configuration files when they run on the server, each application has its own unique copy of these files. Any changes that you make to a connection or configuration file in one application do not affect the files in other applications on the server.

Format of connection pool specification files

XML tag conventions

The following sections describe the XML syntax used to define Host Publisher connection pools, using examples. The following conventions have been used:

- Each file contains a set of tags that correspond to a single instance of the connection or connection pool that the file describes.
- A single top level tag identifies the type of connection being described, such as **<poolconfig>** or **<connconfig>**.
- A tag that describes an instance always has a name attribute.
- Different object-specific tags are used to distinguish the connections that they are instances of. For example, a connection pool specification configuration file can have a **<hodpoolspec>** or a **<dbpoolspec>** tag.
- Within the tag describing the object of interest are nested tags defining that object's properties. Each nested tag within a connection-specific tag is an empty XML tag, and the value of the property that it represents is specified by an attribute.
  - If the property is a "simple" type (integer or string), the value is specified using a value attribute.
  - If the property is a reference to a DbConnSpec, HodConnspec, HodLogonSpec, or LocalUserPool specification, a refname attribute is used to reference that specification's definition. Another file with that name and a fixed extension contains that specification.
  - If the property is a reference to another Java object such as java.util.Properties, whose string representation can be quite big, the value is represented using a nonempty nested tag. An example of this is the sessionprops attribute of the **<hodconnspec>** tag.
  - If the property is a reference to an object that also has an XML representation (such as an HOD macro), then the object is stored in a separate file, and an empty tag with a filename attribute is used to reference that file.
- If a property value is not specified in the XML tag, the default value is used during execution.

  Notes:

  1. All timeout values are integers (32 bit), and the unit of time is seconds.
  2. A timeout value of 0 indicates no waiting. A timeout value of -1 indicates an infinite wait. For counters, the upper limit is always the maximum value of the primitive integer type in Java (2,147,483,647).

3. While all attribute values in XML are strings, type information is provided for each property that the attribute represents since that will limit the string values that can appear (for example, if boolean, valid values are true and false).

XML Tags for connection specifications

A file defines each instance of the ConnSpec record. A connection specification defines how to connect to a data source. Whether connection pooling is used for this definition is defined by the pool specifications. A connection specification is nested within a single **<connconfig>** tag. The **<hodconnspec>** tag is used to describe an HodConnSpec record, and the **<dbconnspec>** tag is used to describe a DbConnSpec record. These records have different sets of properties, and the nested tags used to set their values are described in separate sections.

Host On-Demand connections: The following XML tags correspond to properties of an HodConnSpec record.

connecttimeout
The time, in seconds, that Host Publisher Server will wait while creating a host connection using Host On-Demand APIs, and priming it by running a connect macro.

The value is an integer, either -1 or 1 or greater. The default is 120.

disconnecttimeout
The time, in seconds, that Host Publisher Server will wait while running a disconnect macro and disconnecting a host connection using Host On-Demand APIs.

The value is an integer, either -1 or 1 or greater. The default is 120.

expresslogon
Contains tags that specify whether the connection uses express logon. The nested tags specify connection information for the Digital Certificate Access Server (DCAS) server. The nested tags are:

- elfenabled
- dcasservername
- dcasserverport

sessionprops
Contains Host On-Demand connection properties.

singlelogon
Set this value to true if this connection does not allow a user ID and password to be used for multiple simultaneous sessions. If this value is set to true and a user list is defined for this connection, the user ID/password pairs in that user list are locked when in use to prevent their being used by simultaneous connections. If this value is set to true and a user ID is not available for the user list, the requester for the connection waits the amount of time specified by the **connecttimeout** property.

Set this value to false if this connection allows a user ID and password to be used for multiple simultaneous connections. If this value is set to false and a user list is defined for this connection, the first user ID/password pair in the user list will be reused for each requested connection.

The value is boolean. The default is false.

JDBC connections: The following XML tags correspond to properties of a DbConnSpec record.

connecttimeout

The time, in seconds, that Host Publisher Server will wait to create a database connection using JDBC APIs.

The value is an integer, either -1 or 1 or greater. The default is 120.

drivername

The name of a JDBC driver (class) that can be used by Host Publisher Server to load the driver.

This string value is mandatory.

urlname

This URL must identify the database to which a connection is created.

This string value is mandatory.

Examples: vm3.connspec

```
<?xml version="1.0"?>
<!DOCTYPE connconfig SYSTEM "connconfig.dtd">
  <connconfig>
    <hodconnspec name="vm3">
    <singlelogon value="false"/>
    <sessionprops>
              SSL=false
              fontSize=10
              autoReconnect=false
              OIAVisible=true
              port=23
              autoConnect=false
              TNEnhanced=false
              fontSizeBounded=true
              autoFontSize=false
              codePage=037
              host=ralvm3
              screensize=2
              sessionType=1
              SSLServerAuthentication=false
              LUName=
              codePageKey=KEY_US
    </sessionprops>
    <expresslogon>
              <elfenabled value="true"/>
              <dcasservername value="9.37.52.31"/>
              <dcasserverport value="809">
    </expresslogon/>
    <disconnecttimeout value="120"/>
    </hodconnspec>
  </connconfig>
```

empdb.connspec

```
<?xml version="1.0"?>
<!DOCTYPE connconfig SYSTEM "connconfig.dtd">
  <connconfig>
    <dbconnspec name="empdb">
        <drivername value="com.ibm.db2.jdbcdvr"/>
        <urlname value="jdbc://myserver.ibm.com/employeedb"/>
        <connecttimeout value="60"/>
    </dbconnspec>
  </connconfig>
```

XML tags for pool specifications

A file defines each instance of a PoolSpec record. A pool specification defines whether a connection pool supports connection pooling and defines properties required to support connection pooling. The pool specification is nested within a

single **<poolconfig>** tag. Pool specification values are only used if connection pooling is enabled. See the description of the **<poolingenabled>** tag for more information.

The **<hodpoolspec>** tag is used to describe an HodPoolSpec record, and the **<dbpoolspec>** tag is used to describe an DbPoolSpec record. Both objects have the same set of properties with values defined using the set of nested tags described below:

connecttimeout
The time, in seconds, for which a requester of a connection waits to acquire a connection from the pool if no connections are available.

The value is an integer, either -1 or 0 or greater. The default is 120.

If connecttimeout is set to -1, the requester will wait forever.

dbconnspec
A reference to a DbConnSpec specification in another file. This tag (with different attributes) is used in .connspec files to define DbConnSpec records.

hodconnspec
A reference to a HodConnSpec specification in another file. This tag (with different attributes) is also used in .connspec files to define HodConnSpec records.

hodlogonspec
A reference to a HodLogonSpec specification in another file. This tag (with different attributes) is also used in .logonspec files to define HodLogonSpec records.

localuserpool
A reference to a LocalUserPool specification in another file. This tag (with different attributes) is used in .userpool files to define LocalUserPool records.

maxbusytime
The time in seconds, since a connection was last accessed, after which it is reclaimed. A connection is considered to be accessed when it is acquired or when it is released to Host Publisher Server to save until the next Integration Object in a chained application acquires it. The connection is reclaimed and terminated if maxbusytime is not set to -1 and either of the following occurs:

- An Integration Object acquires a connection, but does not release it to the pool in the time specified by maxbusytime
- An Integration Object releases a connection to Host Publisher Server to save for the next Integration Object in a chain, but the next Integration Object does not acquire it in the time specified by maxbusytime

The value is an integer, either -1 or 60 or greater. The default is -1.

If maxbusytime is set to -1, a busy connection is never reclaimed.

Note: The maxbusytime parameter is in effect even when pooling is not enabled. If an Integration Object acquires a connection that is not pooled, the connection is reclaimed if the connection is not active and maxbusytime is not set to -1.

maxconnections
This is the maximum size of the pool. Once this many connections have been created and all connections have been acquired, the next requester will wait unless **overflowallowed** is set to true. In that case, a new non-pooled connection is created.

The value is an integer. The default is 1.

maxidletime
The time, in seconds, after which a connection that is idle is removed from the pool, if the number of connections in the pool exceeds **minconnections**.

The value is an integer, either -1 or 60 or greater. The default is –1.

If maxidletime is set to -1, an idle connection is never removed from the pool.

minconnections
The number of active connections in the pool below which idle connections are not disconnected, regardless of the value of **maxidletime**. This does not imply that Host Publisher Server will create that many connections during initialization. The pool is populated on demand.

The value is an integer. The default is 0.

overflowallowed
If set to true, when a request is received for a connection and none is available (because the **maxconnections** limit has been reached), a new connection outside the pool is created. When this connection is released, it is ended and discarded.

The value is boolean. The default is false.

poolingenabled
If set to true, connection pooling is enabled and a request to acquire a connection from the pool results in an already-initialized connection being returned to the requester, if one is available. When the requester releases this connection, it is returned to the pool for later use.

If set to false, connection pooling is disabled and a request to acquire a connection from the pool results in a new connection being initialized and returned to the requester. When the requester releases this connection, it is terminated and discarded.

Note: If connection pooling is disabled, the values of the following properties are ignored:

- **maxidletime**
- **connecttimeout**
- **minconnections**
- **maxconnections**
- **overflowallowed**

The value is boolean. The default is true.

Examples: callup.poolspec

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE poolconfig SYSTEM "poolconfig.dtd">
<poolconfig>
  <hodpoolspec name="callup">
      <hodconnspec refname="vm6conn"/>
```

```
      <hodlogonspec refname="vm6"/>
      <localuserpool refname="vm6users"/>
      <maxidletime value="600"/>
      <minconnections value="10"/>
      <maxconnections value="20"/>
      <connecttimeout value="30"/>
      <overflowallowed value="true"/>
  </hodpoolspec>
</poolconfig>
```

puborder.poolspec

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE poolconfig SYSTEM "poolconfig.dtd">
<poolconfig>
  <hodpoolspec name="puborder">
      <hodconnspec  refname="vm6conn"/>
      <hodlogonspec  refname="vm6"/>
      <localuserpool  refname="vm6users"/>
      <connecttimeout value="30">
      <minconnections value="30"/>
      <maxconnections value="40"/>
  </hodpoolspec>
</poolconfig>
```

empdb.poolspec

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE poolconfig SYSTEM "poolconfig.dtd">
<poolconfig>
  <dbpoolspec name="empdb">
    <dbconnspec refname="empdb"/>
    <localuserpool refname="empdbusers"/>
    <connecttimeout value="20">
    <minconnections value="5"/>
    <maxconnections value="10"/>
  </dbpoolspec>
</poolconfig>
```

XML Tags for logon and logoff specifications

A file defines each HodLogonSpec record.The logon specification defines information required by Host On-Demand for logging on and off a connection to a host. This file is only pertinent to Integration Objects created by the Host Access application. The **<hodlogonspec>** tag is used to describe an HodLogonSpec record, and is nested within a single **<logonconfig>** tag. The following XML tags correspond to properties of an HodLogonSpec record.

checkinscreendesc

References a file containing a string representation of a com.ibm.eNetwork.ECL.ECLScreenDesc object that is constructed in Host Publisher Studio. This value is only used when connection pooling is enabled. When a connection is returned to Host Publisher Server, the Server checks the current screen against this screen description. If the current screen and this screen description match, the connection is returned to the pool. If the current screen and this screen description do not match, connection recovery might be initiated.

This string value is mandatory.

logoffmacro

References a file containing the Host On-Demand disconnect macro (in Host On-Demand-defined XML format). A disconnect macro may not be needed if the Integration Object's data macro includes disconnect actions or if certain public domain hosts do not need a disconnect step.

This file reference value is optional.

logonmacro

References a file containing the Host On-Demand connect macro (in Host On-Demand-defined XML format). A connect macro may not be needed if the Host Access Integration Object's data macro includes connect actions or if certain public domain hosts do not need a connect step.

This file reference value is optional.

Example: This is the file vm6.logon. In the example, the file names have been derived from the record name by adding a standard suffix.

```
<?xml version="1.0"?>
<!DOCTYPE logonconfig SYSTEM "logonconfig.dtd">
<logonconfig>
  <hdlogonspec name="vm6">
    <logonmacro filename="vm6_logon.macro"/>
    <logoffmacro filename="vm6_logoff.macro"/>
    <checkinscreendesc value="vm6_checkin.screen"/>
  </hdlogonspec>
</logonconfig>
```

XML Tags for user pool specifications

A user pool file contains a list of user ID/password pairs that are used by a connection pool to make a connection. A file is used to define each LocalUserPool record.

For hosts that allow a user ID/password pair to be used by simultaneous multiple connections (for example, AS/400s and JDBC databases), the user list typically has one entry. If more than one entry is specified for such a connection, Host Publisher Server ignores the other entries when selecting user ID/password pairs for logging on to the connection, because it will always use the first connection.

For hosts that do not allow a user ID/password pair to be used by simultaneous multiple connections (for example, 3270 hosts running VM), the Server manages the user list by locking user ID/password pairs that are currently in use. A subsequent request for a connection uses a userID/password pair that is not locked.

The user pool record can be used to store more than just user IDs and passwords. You can associate other properties with user IDs as well as passwords. For instance, you might have a user ID that requires an additional password to log on to another application as part of the session priming process. In this case, the user pool would contain a list of user ID/password pairs with an additional password property associated with each user ID entry. Each property defined in the user list can be encrypted, except the user ID, and each property can use a different level of encryption.

The **<schema>** tag defines each property that should appear in each entry in the user list, and the encryption level for each property.

The **<localuserpool>** tag describes a LocalUserPool record, and is nested within a single **<userconfig>** tag. Multiple **<entry>** tags are used to define the database entries, one for each user ID, password, and any other properties, using **<property>** tags.

The **<localuserpool>** tag has an optional `session` attribute. If the `session` attribute is present with a value that is not null, at least one property in the user list is strongly encrypted. The value for the `session` attribute is set when a user of Host

Publisher Studio transfers a user list to the server and selects strong encryption. Host Publisher Studio prompts the user for a password to be used for strong encryption. When an application containing a strongly encrypted user list is deployed on the Server, the password the user specified for strong encryption must be defined using Host Publisher Administration. If another user list is created with a property that requires strong encryption, and is to be deployed to the same server, the same password must be specified to encrypt that user list. If weak encryption is used, no password is required.

Examples of User Pool Definitions: vm6users.userpool

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE userconfig SYSTEM "userconfig.dtd">
<userconfig>
  <schema>
      <defineproperty encrypt="0" name="_userid"/>
      <defineproperty encrypt="0" name="app_password"/>
      <defineproperty encrypt="0" name="_password"/>
  </schema>
  <localuserpool name="nm01users">
     <entry  key="vm6Userid01">
        <property name="userid" value="vm6Userid01"/>
        <property name="_password" value="vm6Password01"/>
        <property name="app_password" value="apppw1"/>
     </entry>
     <entry  key="vm6Userid02">
        <property name="userid" value="vm6Userid02"/>
        <property name="_password" value="vm6Password02"/>
        <property name="app_password" value="apppw2"/>
     </entry>
  </localuserpool>
</userconfig>
```

empdbusers.userpool

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE userconfig SYSTEM "userconfig.dtd">
<userconfig>
  <schema>
    <defineproperty encrypt="0" name="_userid"/>
    <defineproperty encrypt="2" name="_password"/>
  </schema>
  <localuserpool name="empdbusers"
    <entry key="UserName5">
      <property name="_userid" value="UserName5"/>
      <property name="_password" value="o0+n5w1W3mhej7KWpb6SEw==/>
    </entry>
    <entry key="UserName4">
      <property name="_userid" value="UserName4"/>
      <property name="_password" value="h57hlX=hMrl13baAuFhk9Q==/>
    </entry>
  </localuserpool>
</userconfig>
```

Macro script files

Macro script files are stored in the .ear file and are transferred to the server.

After a macro script (a .macro file) has been created using the Host Access application, you might want to manually edit it. Manually editing macro scripts should only be performed by advanced users.

Connect and disconnect macros created using Host Access are stored in the *Studio_Install_Dir*\Studio\SessionDefs directory, where *Studio_Install_Dir* is

the Host Publisher installation directory for Host Publisher Studio. Data macros are stored in the `\Studio\IntegrationObjects` directory of the Host Publisher installation directory.

Macro scripts can also be edited on Host Publisher Server as part of the deployed application, although we recommend that you use Host Publisher Studio to edit the macros.

Caution: If the application is redeployed, changes made to the macro scripts on Host Publisher Server will be lost.

Macro editing tips

If you edit any of the macro script files in a non-English environment, you must use a UTF-8 capable editor.

The following sections provide tips for editing macros created by Host Publisher.

Editing extract coordinates in a data macro

When you edit extract coordinates in a data macro, you need to modify the extract coordinates in the Integration Object's .hpi file to match the ones you updated in the macro. After updating the .hpi file, use the Host Access application to re-generate the Integration Object. If the extract coordinates in the data macro do not match those in the .hpi file when the Integration Object runs, the data macro extracts the data based on the macro's updated coordinates, but the Integration Object returns the data to your Web application based on the old coordinates. The data in the resulting Web page might be incorrect.

Editing special characters

Host Access uses Host On-Demand to establish a connection to the host. Host On-Demand uses special characters to identify host aid keys. If you want Host On-Demand to interpret text literally instead of treating it as a potential host aid key, you must edit special characters manually in the Host Publisher macro.

For example, the square bracket ([ ) signifies a host aid key such as [enter]. To send the square bracket as a non-special key, input a double square bracket ([[ ) in a Host Publisher macro. While playing back the macro, set the value for xlatehostkeys to true in the prompt.

If you edit the macro in Host Access after making manual changes to the macro, you lose your manual changes. This applies to the prompt tag and numerous other tags. See "Macro script syntax" on page 66 for more information.

The following is an example of a correct prompt:

```
<prompt name="ID" row="1" col="1" default="[some more text" xlatehostkeys="false" />
```

In the following example, [enter] is not translated into the Enter key.

```
<prompt name="ID" row="1" col="1" default="[some more text[enter]" xlatehostkeys="false" />
```

In this case, you should split the prompt:

```
<prompt name="ID" row="1" col="1" default="[some more text" xlatehostkeys="false" />
<input value="[enter]" row="0" col="0" movecursor="true" xlatehostkeys="true" />
```

h

Macro script syntax

This section is excerpted from the IBM WebSphere Host On-Demand *Host Access Beans for Java Reference*. The complete book is included in the Host On-Demand Host Access Toolkit.

Information has been added or modified for usage of macro scripts within Host Publisher.

Introduction

IBM Host On-Demand uses XML because a macro is better suited to the state machine model (the main reason for the move: XML is tailor made for a state machine).

The idea of a state machine may be fairly new to you. The idea behind a state machine, especially in the IBM Host On-Demand macro context, is simple. Think of how you use a host system from a terminal or a terminal emulator (like IBM Host On-Demand). The process you follow when you interact with a host system is illustrated in these steps:

1. The host sends an expected screen down to you at your terminal.
2. You look at and understand which screen is presented to you.
3. You take the required actions based on your understanding (type keystrokes, and so forth).
4. Another screen is presented after these actions.
5. If you see the screen you expected, repeat steps 2, 3, and 4.
6. If you do not see the screen you expected, call the help desk or handle the error.

This is the idea behind a state machine in the Macro context (although the Macro can't call the help desk for you). The states are the screens you expect to see, and you take actions on those screens to change from one state, or screen, to another. That's it, see a screen, perform the action, see the next screen. It is easier to understand (and program) a macro with this approach than having several if-then-else and do-while programming statements. Remember, see a screen, perform the action, see the next screen.

Now take a look at how well suited XML is to coding a macro. Here is an example of how to specify a connect macro:

```
<HAScript name="" description="" timeout="60000" pausetime="200" promptall="false"
          author="" creationdate="" supressclearevents="false" >
  <screen name="Screen1.1" entryscreen="true" exitscreen="false" transient="false">
     <description uselogic="1 and ((4 and 5) OR (2 or 3))" >
       <oia status="NOTINHIBITED" optional="false" invertmatch="false" />
       <block row="20" col="2" casesense="true" optional="true" invertmatch="false" >
         <string value="USERID   ===>" />
         <string value="PASSWORD ===>" />
       </block>
       <string value="USERID" row="15" col="1" casesense="true" optional="true"
               invertmatch="false" />
       <numinputfields number="16" optional="false" invertmatch="false" />
       <cursor row="20" col="16" optional="false" invertmatch="false" />
     </description>
     <actions>
         <input value="myID" row="0" col="0" movecursor="true" xlatehostkeys="true"
                encrypted="false" />
         <input value="myPW" row="0" col="0" movecursor="true" xlatehostkeys="true"
                encrypted="true" />
         <input value="[enter]" row="0" col="0" movecursor="true" xlatehostkeys="true"
```

```
                encrypted="false" />
     </actions>
     <nextscreens timeout="0" >
          <nextscreen name="Screen2" />
     </nextscreens>
  </screen>
  <screen name="Screen2" entryscreen="false" exitscreen="false" transient="false">
     <description uselogic="1 and 2" >
       <oia status="NOTINHIBITED" optional="false" invertmatch="false" />
       <string value="MORE..." row="20" col="20" casesense="true" optional="false"
               invertmatch="false" />
     </description>
     <actions>
         <input value="[clear]" row="0" col="0" movecursor="true" xlatehostkeys="true"
                encrypted="false" />
     </actions>
     <nextscreens timeout="0" >
          <nextscreen name="Screen3.1" />
     </nextscreens>
  </screen>
  <screen name="Screen3.1" entryscreen="false" exitscreen="true" transient="false">
     <comment>
      This screen description defines the connection pool checkin screen.
     </comment>
     <description uselogic="1 and (2 and 3)" >
       <oia status="NOTINHIBITED" optional="false" invertmatch="false" />
       <string value="Ready;" row="1" col="1" casesense="true" optional="false"
               invertmatch="false" />
       <string value="Ready;" row="2" col="1" casesense="true" optional="false"
               invertmatch="true" />
     </description>
     <actions>
     </actions>
     <nextscreens timeout="0" >
     </nextscreens>
  </screen>
<HAScript>
```

These lines of code demonstrate the power of this syntax. All the screens you expect to see for a task (like connecting) are coded within **<screen>** tags in XML. You describe the screen in a **<description>** tag, specify the actions for the screen in an **<actions>** tag, and specify the screen you want to see next in a **<nextscreens>** tag.

Keep in mind that the actions happen in sequence. The **<screen>** tag describes a logon screen with the text **USERID** and **PASSWORD** on the screen and the screen's cursor position at row 20, column 10. If the macro logic sees a screen matching this description, it prompts the user for an ID and password, places the prompt results at the specified row and column positions, and sends the ENTER key, effectively connecting the user to the host. The **<nextscreens>** tag specifies the name of another **<screen>** tag that appears later in the macro. If the next screen does not appear, the macro logic returns an error.

Although there are a large number of valid XML tags, XML is not complicated. A screen is specified with a description, actions, and the next screens. When a macro is played and a screen matching the description appears, the actions are executed for that screen and the macro logic monitors the host for any next screens specified.

Macro Syntax

The following details each valid macro tag:

```
<HAScript>
  <screen>
    <comment>
    <description>
      <oia>
      <cursor>
      <numfields>
      <numinputfields>
      <string>
      <block>
      <attrib>
      <customreco>
    <actions>
      <prompt>
      <input>
      <extract>
      <message>
      <trace>
      <xfer>
      <pause>
      <mouseclick>
      <boxselect>
      <commwait>
      <custom>
    <nextscreens>
      <nextscreen>
    <recolimit>
```

The following XML tags and their attributes are valid in the IBM Host On-Demand Macro XML namespace. This description of the tags is structured like an actual macro file.

Note: The tag and attribute values are not case sensitive.

Attention: All characters in a macro must be Unicode characters. Most text editors support this by default, because they use the ASCII character set, which is at the lower end of the Unicode character set.

<HAScript> tag: The **<HAScript>** tag is the main enclosing tag for the macro. All other tags at this level that are not HAScript are ignored by the parser.

The attributes of the **<HAScript>** tag are:

name The name of the macro. This attribute is optional. The name can contain any valid Unicode character.

description
The description of the macro. This attribute is optional. The description can contain any valid Unicode character.

author The creator of the macro. This attribute is optional. The author can contain any valid Unicode character.

creationdate
The date the macro was created. This attribute is optional. The creationdate can contain any valid Unicode character. The date format is not checked.

promptall
This launches all prompts at the beginning of the macro. This attribute is optional. The default is true. The value must be true or false.

pausetime
The sleep time in milliseconds initiated after a screen is matched. This is used to let the host quiet down. This attribute is optional. The default is no

pause. The value must be a number. The default is 300 milliseconds. If a <pause> tag is specified for a specific screen, the value specified on the <pause> tag overrides this value.

Note: The maximum pause time is limited to the platform on which the macro is running.

timeout
The allowable time in milliseconds between recognition events. If time expires, the macro goes into the error state. You can override this value in the <nextscreens> tag. The value must be a number. The default is 60,000 milliseconds (60 seconds).

Note: The maximum pause time is limited to the largest numeric value supported on the platform on which the macro is running.

suppressclearevents
This is an advanced feature that determines whether the system should ignore screen events when a host application sends a clear screen command immediately followed by an end of record indicator in the data stream. You may want to set this value to true if you have screens in your application flow that have all blanks in them. If there is a valid blank screen in the macro and clear commands are not ignored, it is possible that a screen event with all blanks will be generated by clear commands coming from an ill-behaved host application. This will cause a screen recognition event to be processed and the valid blank screen will match when it shouldn't have matched. This attribute is optional. The default is false. The value must be true or false.

Example:

```
<HAScript name="Logon Macro" description="Logs me on" author="btwebb"
     creationdate="12/29/1998" promptall="true" pausetime="500" timeout="10000" >
 ...
</HAScript>
```

<screen> tag: The **<screen>** tag is the enclosing tag for the screen.

The attributes of the **<screen>** tag are:

name The unique identifier for the screen. This attribute is mandatory and **must be a unique string among the other screen IDs**. The name can contain any valid Unicode character.

entryscreen
If true, the screen should be the first screen seen. Any other screen generates an error. This value must be true or false. This attribute is optional. The default is false.

Note: There can be only one screen with the entryscreen attribute set to true.

exitscreen
If true, a match on the screen causes the macro to stop playing. You can have multiple screens with the exitscreen attribute set to true. This value must be true or false. This attribute is optional. The default is false.

transient
If true, the screen is handled as transient. Transient screens exist outside the normal macro flow. **They are matched after nontransient screens. If**

you specify next screens in a transient screen, the next screens are ignored. Use this attribute to specify a screen that can appear at any time in the screen flow. This value must be true or false. This attribute is optional. The default is false.

pause Time (in milliseconds) to pause before the screen recognition engine attempts to match next screens. A value greater or equal to 0 overrides the value specified on the pausetime attribute of the <HAScript> tag.. The default value is -1.

Example:

```
<screen name="screen1" entryscreen="true" exitscreen="false" transient="false">
...
</screen>
```

<comment> tag: The **<comment>** tag for the screen. This can contain any valid Unicode character.

There are no attributes for the **<comment>** tag.

Example:

```
<comment> ... </comment>
```

<description> tag: The **<description>** tag is the enclosing tag for the description associated with the screen.

The attributes of the **<description>** tag are:

uselogic

Determines the boolean logic for screen recognition. The numbers in the value represent the sequential positions of the descriptors in the **<description>** tag. There must be a descriptor for each number in the value.

In a macro generated by Host Access, the first number in the value always represents the position of the **<oia>** tag.

Notes:

1. On the Host Publisher Server machine, if the uselogic attribute is present, the optional and invertmatch attributes of the descriptors are ignored.
2. On the Host Publisher Studio machine, if you change any of the following:
   - The value of the uselogic attribute of the **<description>** tag
   - The values of the optional and invertmatch attributes of the descriptors
   - The positions of the descriptors

   and you load the macro into Host Access, the value of the uselogic attribute is overwritten to match the positions and optional and invertmatch attributes of the descriptors.
3. If you make any of the changes listed above, and the value of the uselogic attribute contains a number for a descriptor that does not exist, the following problems occur:
   - On Host Publisher Server, the macro fails when Host On-Demand attempts to load it, and an error message is logged.

- In Host Access, the macro does not load when you open the Integration Object, and an error message is issued. The macro appears blank in the macro tree. Other macros are not affected.

4. If you have a syntax error in the value of the uselogic attribute, such as unmatched parentheses, the following problems occur:
   - On Host Publisher Server, the macro fails when Host On-Demand attempts to load it, and an error message is logged.
   - In Host Access, the macro does not load when you open the Integration Object, and an error message is issued. The macro appears blank in the macro tree. Other macros are not affected.

Example:

```
<description uselogic="1 and (2 or !3)"> ... </description>
```

<oia> tag: The **<oia>** tag specifies an operator information area (OIA) condition to match. This tag is optional. The default is to wait for inhibit status.

The attributes of the **<oia>** tag are:

status If NOTINHIBITED, the OIA must be uninhibited for a match to occur. If DONTCARE, the OIA inhibit status is ignored. This has the same effect as not specifying OIA at all. Valid values are NOTINHIBITED and DONTCARE. This is a required attribute.

optional
If false, this descriptor is considered non-optional during screen recognition. If the descriptors are comprised of more than one non-optional descriptor, and more than one optional descriptor, the non-optional descriptors are checked first. If all of the non-optional descriptors match, the screen matches. If at least one of the non-optional descriptors does not match, the optional descriptors are checked. One of the optional descriptors must match for the screen to match. Otherwise, the screen fails to match. The value must be true or false. This attribute is optional. The default is false.

Note: If the uselogic attribute is specified on the <description> tag, this attribute is ignored.

invertmatch
If true, recognition matching passes only if the screen does not match this description element (logical NOT not operation). The value must be true or false. This attribute is optional. The default is false.

Note: If the uselogic attribute is specified on the <description> tag, this attribute is ignored.

Example:

```
<oia status="NOTINHIBITED" optional="false" invertmatch="false" />
```

<cursor> tag: The **<cursor>** tag describes the screen based on the position of the cursor.

The attributes of the **<cursor>** tag are:

row The row position of the cursor. The value must be a number. This is a required attribute.

col The column position of the cursor. The value must be a number. This is a required attribute.

optional
If false, this descriptor is considered non-optional during screen recognition. If the descriptors are comprised of more than one non-optional descriptor, and more than one optional descriptor, the non-optional descriptors are checked first. If all of the non-optional descriptors match, the screen matches. If at least one of the non-optional descriptors does not match, the optional descriptors are checked. One of the optional descriptors must match for the screen to match. Otherwise, the screen fails to match. The value must be true or false. This attribute is optional. The default is false.

Note: If the uselogic attribute is specified on the <description>, this attribute is ignored.

invertmatch
If true, recognition matching passes only if the screen does not match this description element (boolean not operation). The value must be true or false. This attribute is optional. The default is false.

Note: If the uselogic attribute is specified on the <description> tag, this attribute is ignored.

Example:

```
<cursor row="1" col="1" optional="false" invertmatch="false" />
```

<numfields> tag: The **<numfields>** tag defines the total number of fields on the screen. This tag is optional. The number of fields not used if not specified.

The attributes of the **<numfields>** tag are:

number
The field count. The value must be a number. This is a required attribute.

optional
If false, this descriptor is considered non-optional during screen recognition. If the descriptors are comprised of more than one non-optional descriptor, and more than one optional descriptor, the non-optional descriptors are checked first. If all of the non-optional descriptors match, the screen matches. If at least one of the non-optional descriptors does not match, the optional descriptors are checked. One of the optional descriptors must match for the screen to match. Otherwise, the screen fails to match. The value must be true or false. This attribute is optional. The default is false.

Note: If the uselogic attribute is specified on the <description>, this attribute is ignored.

invertmatch
If true, recognition matching passes only if the screen does not match this description element (boolean not operation). The value must be true or false. This attribute is optional. The default is false.

Note: If the uselogic attribute is specified on the <description> tag, this attribute is ignored.

Example:

```
<numfields number="10" optional="false" invertmatch="false" />
```

<numinputfields> tag: The **<numinputfields>** tag defines the total number of input fields on the screen. This tag is optional. The number of input fields is not used if not specified.

The attributes of the **<numinputfields>** tag are:

number
The field count. The value must be a number. This is a required attribute.

optional
If false, this descriptor is considered non-optional during screen recognition. If the descriptors are comprised of more than one non-optional descriptor, and more than one optional descriptor, the non-optional descriptors are checked first. If all of the non-optional descriptors match, the screen matches. If at least one of the non-optional descriptors does not match, the optional descriptors are checked. One of the optional descriptors must match for the screen to match. Otherwise, the screen fails to match. The value must be true or false. This attribute is optional. The default is false.

Note: If the uselogic attribute is specified on the <description>, this attribute is ignored.

invertmatch
If true, recognition matching passes only if the screen does not match this description element (boolean not operation). The value must be true or false. This attribute is optional. The default is false.

Note: If the uselogic attribute is specified on the <description> tag, this attribute is ignored.

Example:

```
<numinputfields number="10" optional="false" invertmatch="false" />
```

<string> tag: The **<string>** tag describes the screen based on a string.

The attributes of the **<string>** tag are:

value The string value. This value can contain any valid Unicode character. This is a required attribute.

row The starting row position for a string at an absolute position or in a rectangle. The value must be a number. This value is optional. If not specified, Macro logic searches the entire screen for the string. If specified, col position is required. <erow> and <ecol> attributes can also be specified to specify a string in a rectangular area.

Note: Negative values are valid and are used to indicate relative position for the bottom of the screen (for example, -1 is the last row).

col The starting column position for the string at an absolute position or in a rectangle. The value must be a number. This attribute is optional.

erow The ending row position for string in a rectangle. The value must be a number. This attribute is optional. If both erow and ecol are specified, string is in a rectangle.

ecol The ending column position for string in a rectangle. The value must be a number. This attribute is optional. If both erow and ecol are specified, string is in a rectangle.

casesense
If true, string comparison is case sensitive. The value must be true or false. This attribute is optional. The default is false.

optional
If false, this descriptor is considered non-optional during screen recognition. If the descriptors are comprised of more than one non-optional descriptor, and more than one optional descriptor, the non-optional descriptors are checked first. If all of the non-optional descriptors match, the screen matches. If at least one of the non-optional descriptors does not match, the optional descriptors are checked. One of the optional descriptors must match for the screen to match. Otherwise, the screen fails to match. The value must be true or false. This attribute is optional. The default is false.

Note: If the uselogic attribute is specified on the <description>, this attribute is ignored.

invertmatch
If true, recognition matching passes only if the screen does not match this description element (boolean not operation). The value must be true or false. This attribute is optional. The default is false.

Note: If the uselogic attribute is specified on the <description> tag, this attribute is ignored.

Examples:

```
<string value="hello" row="1" col="1" optional="false" invertmatch="false" />
<string value="hello" row="1" col="1" erow="11" ecol="11" casesense="false"
          optional="false" invertmatch="false" />
<string value="hello" />
```

<block> tag: The **<block>** tag describes the screen based on a group of strings. This tag is used instead of multiple **<string>** tags, when the strings are positioned on the screen relative to the first string. The contents of **<block>** tag are considered a single descriptor.

The attributes of the **<block>** tag are the same as the **<string>** tag, with the exception of the value attribute.

Example:

```
<block row="20" col="2" casesense="true" optional="true" invertmatch="false" >
  <string value="USERID   ===>" />
  <string value="PASSWORD ===>" />
</block>
```

<attrib> tag: The **<attrib>** tag describes the screen based on an attribute. This is an advanced feature and should only be used if needed. Usually all the other description elements are enough to describe a screen.

The attributes of the **<attrib>** tag are:

plane The plane value string that the attribute resides in. Valid values are COLOR_PLANE, FIELD_PLANE, and EXFIELD_PLANE. This is a required attribute.

value The hex value string of the attribute. Example, value=″0xA0″. This is a required attribute.

row The row position of the attribute. The value must be a number. This is a required attribute.

col The column position of the attribute. The value must be a number. This is a required attribute.

optional
If false, this descriptor is considered non-optional during screen recognition. If the descriptors are comprised of more than one non-optional descriptor, and more than one optional descriptor, the non-optional descriptors are checked first. If all of the non-optional descriptors match, the screen matches. If at least one of the non-optional descriptors does not match, the optional descriptors are checked. One of the optional descriptors must match for the screen to match. Otherwise, the screen fails to match. The value must be true or false. This attribute is optional. The default is false.

Note: If the uselogic attribute is specified on the <description> tag, this attribute is ignored.

invertmatch
If true, recognition matching passes only if the screen does not match this description element (boolean not operation). The value must be true or false. This attribute is optional. The default is false.

Note: If the uselogic attribute is specified on the <description> tag, this attribute is ignored.

Example:

```
<attrib value="0x01" row="1" col="1" plane="COLOR_PLANE" optional="false"
          invertmatch="false" />
```

<customreco> tag: The macro logic will call out to any custom recognition listeners for the custom tag to have the listener do its own custom screen recognition logic.

The attributes of the **<customreco>** tag are:

ID The unique identifier for the custom description element. Allows for multiple custom elements. This can be any valid Unicode character. This is a required attribute.

optional
If false, this descriptor is considered non-optional during screen recognition. If the descriptors are comprised of more than one non-optional descriptor, and more than one optional descriptor, the non-optional descriptors are checked first. If all of the non-optional descriptors match, the screen matches. If at least one of the non-optional descriptors does not match, the optional descriptors are checked. One of the optional descriptors must match for the screen to match. Otherwise, the screen fails to match. The value must be true or false. This attribute is optional. The default is false.

Note: If the uselogic attribute is specified on the <description> tag, this attribute is ignored.

invertmatch
If true, recognition matching passes only if the screen does not match this description element (boolean not operation). The value must be true or false. This attribute is optional. The default is false.

Note: If the uselogic attribute is specified on the <description> tag, this attribute is ignored.

Example:

```
<customreco id="id1" optional="false" invertmatch="false"/>
```

<actions> tag: The **<actions>** tag is the enclosing tag for the actions associated with the screen.

The attributes of the **<actions>** tag are:

promptall
If this value is set to true, the Macro bean will gather all prompts **within the current action tag** and launch them as one prompt event. The value must be true or false. This attribute is optional. The default is false.

Example:

```
<actions promptall="true"> ... <actions>
```

<prompt> tag: The **<prompt>** tag specifies a prompt to be handled for the screen.

The attributes of the **<prompt>** tag are:

row The row to place the prompt. The value must be a number. This is a required attribute.

col The column to place the prompt. The value must be a number. This is a required attribute.

len The length of the prompt. The value must be a number. This is a required attribute.

name The name of the prompt. This can be any valid Unicode character. This attribute is optional.

description
The description of the prompt. This can be any valid Unicode character. This attribute is optional.

default
The prompt's default value. This can be any valid Unicode character. This attribute is optional.

clearfield
This clears the host field on placement of prompt text. The value must be true or false. This attribute is optional. The default is false.

encrypted
Use a password echo character. The value must be true or false. This attribute is optional. The default is false.

xlatehostkeys
If true, host key mnemonics (example, [enter]) will be translated. For a list of key mnemonics, see the Host On-Demand online help. The value must be true or false. This attribute is optional. The default is false. If you do not have this value set to true (which is normal because you wouldn't ask

users to type key mnemonics), don't forget to code an input tag after the prompt(s) for the current actions to get the prompt data entered onto the host.

Example:

```
<prompt name="ID" row="1" col="1" len="8" description="ID for Logon"
          default="btwebb" clearfield="true" encrypted="true"/>
```

<input> tag: The **<input>** tag specifies keystrokes to be placed on the screen.

The attributes of the **<input>** tag are:

row The row position to send the keys. The value must be a number. This attribute is optional. This defaults to current cursor position.

col The column position to send the keys. The value must be a number. This attribute is optional. This defaults to current cursor position.

movecursor
Whether to place the cursor at the end of the input string. The value must be true or false. This attribute is optional. This defaults to false.

value The text that is sent to the screen. This can be any valid Unicode character. This is a required attribute.

xlatehostkeys
If true, host key mnemonics (example, [enter]) will be translated. For a list of key mnemonics, see the Host On-Demand online help. The value must be true or false. This attribute is optional. The default is true.

encrypted
If true, keystrokes that are typed are not displayed on the screen. The value must be true or false. This attribute is optional. The default is false.

Example:

```
<input value="[clear]" row="0" col="0" movecursor="true" xlatehostkeys="true"
       encrypted="false" />
```

<extract> tag: The **<extract>** tag specifies an extract to be handled for the screen.

The attributes of the **<extract>** tag are:

name The name of the extract. This can be any valid Unicode character. This attribute is optional.

srow Upper left row of the bounding extract rectangle. The value must be a number. This is a required attribute.

scol The upper left column of the bounding extract rectangle. The value must be a number. This is a required attribute.

erow The lower right row of the bounding extract rectangle. The value must be a number. This is a required attribute.

ecol The lower right column of the bounding extract rectangle. The value must be a number. This is a required attribute.

Example:

```
<extract name="Get Data" srow="1" scol="1" erow="11" ecol="11" />
```

<message> tag: The **<message>** tag specifies a message to be sent to the user.

The attributes of the **<message>** tag are:

title The title to display in the message dialog. This can be any valid Unicode character. This attribute is optional. This defaults to macro name.

value The message to display in the dialog. This can be any valid Unicode character. This is a required attribute.

Example:

```
<message value="yourvalue" title="YourMessage" />
```

Note: To aid in debugging Host Publisher macro execution, you can set distinctive messages in connect, data, and disconnect macros. During macro execution, the macroMessage Integration Object property is set to the last message tag executed. JSPs can query the value of the last message encountered using the Host Publisher getHPubMacroMessage method. For information on the getHPubMacroMessage method and other Integration Object methods that you can use in your WebSphere applications, see "Integration Object methods" on page 11.

<trace> tag: The **<trace>** tag specifies a string to be sent to one of several trace facilities.

The attributes of the **<trace>** tag are:

type The type can either be sent to the IBM Host On-Demand trace facility, a user trace event, or to the command line. Respectively, the types are HODTRACE, USER, and SYSOUT. This is a required attribute.

Note: To aid in debugging and controlling Host Publisher macro execution, set the type attribute to **USER**.

value The text that is sent to trace. This can be any valid Unicode character. This is a required attribute.

Example:

```
<trace value="hello" type="HODTRACE" />
```

<xfer> tag: The **<xfer>** tag transfers a file to or from a host system.

The attributes of the **<xfer>** tag are:

direction
The direction for the file transfer. The allowable types are SEND (file from PC to host) and RECEIVE (file from host to PC). This is a required attribute.

pcfile The PC file name to be used for the file transfer. This should point to a valid file on your system. This is a required attribute.

hostfile
The host file name to be used for the file transfer. This should point to a valid file on your host system. This is a required attribute.

clear Indicates whether the Macro bean should clear the host screen before performing the file transfer. The value must be true or false. This attribute is optional.

timeout
Sets the time out value (in milliseconds) for the file transfer. If the transfer

does not complete in this given time, the macro will end in error. The value must be a number in milliseconds. This attribute is optional and the default is 10000 milliseconds or 10 seconds.

options
Sets the host specific options for the file transfer. Options are different for every type of host system. See the file transfer bean documentation or contact your host system administrator for valid options for your host system. This value must be a Unicode string. This attribute is optional and the default is no options.

pccodepage
Sets the PC code page to use in the file transfer. The value must be a valid PC code page. See the Host On-Demand online help for session configuration for valid code page values. This attribute is optional.

hostorientation
Sets the host character orientation to use in the file transfer. This applies to BIDI (bidirectional) environments only. See the Host On-Demand online help for session configuration for valid values. This attribute is optional and defaults to no value.

pcorientation
Sets the PC character orientation to use in the file transfer. This applies to BIDI (bidirectional) environments only. See the Host On-Demand online help for session configuration for valid values. This attribute is optional and defaults to no value.

pcfiletype
Sets the PC file type to use in the file transfer. This applies to BIDI (bidirectional) environments only. See the Host On-Demand online help for session configuration for valid values. This attribute is optional and defaults to no value.

lamalefexpansion
Sets whether Lam Alef expansion will be used in the file transfer. This applies to BIDI (bidirectional) environments only. See the Host On-Demand online help for session configuration for page values. This attribute is optional and defaults to no value.

lamalefcompression
Sets whether Lam Alef compression will be used in the file transfer. This applies to BIDI (bidirectional) environments only. See the Host On-Demand online help for session configuration for page values. This attribute is optional and defaults to no value.

Example:

```
<xfer direction="send" pcfile="c:\myfile.txt" hostfile="myfile text A0" />
```

<pause> tag: The **<pause>** tag causes the macro engine to sleep for the number of milliseconds specified. This action is useful for pausing between several file transfers. The value specified for the <pause> tag overrides the value specified on the pausetime attribute of the <HAScript> tag.

The attributes of the **<pause>** tag are:

value The time to pause. The value must be a number (in milliseconds). This attribute is optional. The default is 10000 milliseconds or 10 seconds.

Example:

```
<pause value="2000" />
```

<mouseclick> tag: The **<mouseclick>** tag simulates a user mouse click on the Terminal bean. This essentially sets the cursor at a given row and column position.

The attributes of the **<mouseclick>** tag are:

row The host screen row position for the mouse click. This must be a number within the host screen coordinate system (example, 24 rows by 80 columns). This is an optional attribute and the default value is 1.

col The host screen column position for the mouse click. This must be a number within the host screen coordinate system (example, 24 rows by 80 columns). This is an optional attribute and the default value is 1.

Example:

```
<mouseclick row="20" col="16" />
```

<boxselect> tag: The **<boxselect>** tag is used for either marking or unmarking the marking rectangle on the Terminal bean.

The attributes of the **<boxselect>** tag are:

srow The upper left row of the bounding selection rectangle. The value must be a number within the host screen coordinate system (example, 24 rows by 80 columns). Negative values are allowed and specify a virtual position from the last row (for example, if the Screen has 24 rows, a row value of -2 points to the 22nd row). This is a required attribute.

scol The upper left column of the bounding selection rectangle. The value must be a number within the host screen coordinate system. Negative values are allowed and specify a virtual position from the last column. This is a required attribute.

erow The lower right row of the bounding selection rectangle. The value must be a number within the host screen coordinate system. Negative values are allowed and specify a virtual position from the last row. This is a required attribute.

ecol The lower right column of the bounding selection rectangle. The value must be a number within the host screen coordinate system. Negative values are allowed and specify a virtual position from the last column. This is a required attribute.

type The type of selection action to perform. The value must be either SELECT or DESELECT. This is an optional attribute and the default is SELECT.

Example:

```
<boxselect srow="1" scol="1" erow="11" ecol="11" type="SELECT" />
```

<commwait> tag: The **<commwait>** tag is used for performing a communication status wait during a macro's execution.

The attributes of the **<commwait>** tag are:

value The type of communication status to wait for. Valid values are CONNECTION_INIT, CONNECTION_PND_INACTIVE, CONNECTION_INACTIVE, CONNECTION_PND_ACTIVE, CONNECTION_ACTIVE, CONNECTION_READY, and CONNECTION_DEVICE_NAME_READY. The meaning of these types is

documented in the Java documentation for the ECLConnection object in the Host On-Demand Host Acceess Class Library documentation. The two most used and most meaningful types are CONNECTION_READY and CONNECTION_INACTIVE. This is a required attribute.

timeout
Sets the time out value (in milliseconds) for the communication wait. If the wait does not complete in this given time, the macro will end in error. The value must be a number in milliseconds. This attribute is optional and the default is no time out.

Example:

```
<commwait value="CONNECTION_READY" timeout="10000" />
```

<custom> tag: The **<custom>** tag enables the user to have an exit to Java code. See the Host On-Demand Java documentation for the MacroActionCustom class.

The attributes of the **<custom>** tag are:

id The ID of the callout code that the Macro bean will use. This can be any valid Unicode character. This is a required attribute.

args The argument string that can be passed to the callout. This can be any valid Unicode character. This attribute is optional.

Example:

```
<custom id="custom1" args="YourArgument" />
```

<nextscreens> tag: The **<nextscreens>** tag contains all the valid next screens to be recognized after the current screen's actions have been executed.

The attributes of the **<nextscreens>** tag are:

timeout
The allowable time in milliseconds that can elapse between current screen and any next screen before the macro bean will go into the error state. This overrides the timeout attribute for the entire macro. The value must be a number. This attribute is optional. The default is to use the overall macro timeout.

Example:

```
<nextscreens> ... </nextscreens>
```

<nextscreen> tag: The **<nextscreen>** tag forces a next screen. Multiple **<nextscreen>** tags are allowed. If a screen appears that is in the macro but is not a next screen, the macro will go into an error state. If the next screen refers to a screen tag that doesn't exist, the macro will have a parse error.

The attributes of the **<nextscreen>** tag are:

name The name of the <screen> element that is the valid next screen. This can be any valid Unicode character. This is a required attribute.

Example:

```
<nextscreen name="screen1" />
```

<recolimit> tag: The **<recolimit>** tag is for advanced use only. It is used to enforce a limited amount of time a screen can be recognized **in a row** before it

goes to the screen indicated in the goto attribute. This tag is useful for screen looping where you know exactly how many times you'll see a given screen in a row. It also is a safeguard against infinite screen recognition.

The attributes of the **<recolimit>** tag are:

value The allowable number of times to recognize a screen. This value must be a number. This is a required attribute.

Note: The actions will not be executed the last time the screen is recognized.

goto The name of the screen to go to when recognition limit has been reached. This can be any valid Unicode character but the screen must exist in the macro. For Host Publisher, this attribute is required. If no goto screen is given, the macro terminates.

Example:

```
<recolimit value="3" goto="endscreen"/>
```

Appendix A. Notices

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:
INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

© Copyright IBM Corp. 2000, 2002

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
TL3B/062
3039 Cornwallis Road
RTP, NC 27709-2195
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Programming interface information

This User's Guide contains information on intended programming interfaces that allow the customer to write programs to obtain the services of Host Publisher.

Appendix B. Trademarks

The following terms are trademarks of the IBM Corporation in the United States or other countries or both:

- AIX
- DB2 Universal Database
- IBM
- OS/400
- WebSphere

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and FrontPage are trademarks or registered trademarks of Microsoft Corporation in the United States, other countries, or both.

Other company, product, and service names may be trademarks or service marks of others.

© Copyright IBM Corp. 2000, 2002

Index

Special Characters

A

© Copyright IBM Corp. 2000, 2002

B

C

D

E

F

G

H

I

J

L

M

N

O

P

R

S

T

U

V

W

X

Readers' Comments — We'd Like to Hear from You

IBM® WebSphere® Host Publisher
Programmer's Guide and Reference
Version 4.0

Overall, how satisfied are you with the information in this book?

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Overall satisfaction	☐	☐	☐	☐	☐

How satisfied are you that the information in this book is:

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Accurate	☐	☐	☐	☐	☐
Complete	☐	☐	☐	☐	☐
Easy to find	☐	☐	☐	☐	☐
Easy to understand	☐	☐	☐	☐	☐
Well organized	☐	☐	☐	☐	☐
Applicable to your tasks	☐	☐	☐	☐	☐

Please tell us how we can improve this book:

Thank you for your responses. May we contact you? ☐ Yes ☐ No

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you.

Name _____ Address _____

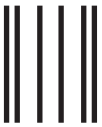
Company or Organization _____

Phone No. _____

Readers' Comments — We'd Like to Hear from You

Cut or Fold
Along Line

Fold and Tape **Please do not staple** Fold and Tape



NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST-CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

IBM Corporation
Software Reengineering
Department G7IA/ Bldg 503
Research Triangle Park, NC
27709-9990

Fold and Tape **Please do not staple** Fold and Tape

Cut or Fold
Along Line

Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.