IBM

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

Version 2 Release 2

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

Version 2 Release 2

Note

Before using this information and the product it supports, be sure to read the general information under "Appendix A. Notices" on page 55.

First Edition (July 2000)

Contents

About this information v
For more information v
Information on the Web vi

Chapter 1. Programming with IBM® Host Publisher Integration Objects 1
Preparing an Integration Object to run in an IDE or with a servlet 1
Setting up VisualAge for Java to run an Integration Object 2
Setting up Symantec Visual Café to run an Integration Object. 2
Writing an application to invoke an Integration Object 3
Writing a servlet to invoke an Integration Object . . 5
Sample code for initializing and starting the runtime 7

Chapter 2. Using the XML Legacy Gateway 9
The xmlLegacyGateway servlet. 10
The xmlAppData JavaBean 11
The HostConnection JavaBean 12

Chapter 3. Using Remote Integration Objects 13
Creating Remote Integration Objects 13
Remote Integration Object files 13
Obtaining Integration Object data in XML format. . 14

Chapter 4. Host Publisher File formats 17
Integration Object project (.hpi) file 17
Host Publisher application (.hpa) file 20
Integration Object source (.java) file 21
JavaServer Pages (JSP) Web page files 21
Connection and configuration files 26
Format of connection pool specification files . . 27
The application manifest file. 34
XML Tags for the application manifest 35
Macro script files 37
Macro editing tips 37
Macro script syntax. 37
The server.properties file 49

Appendix A. Notices 55
Programming interface information 56

Appendix B. Trademarks 57

Index 59

About this information

This information is designed to help you, the Host Publisher programmer, understand how to write applications and servlets to invoke Host Publisher Integration Objects in an integrated development environment (IDE), and how to use the XML Legacy Gateway and Remote Integration Object 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 "For more information").

Note: Consult the product README and the Host Publisher Web site, http://www.ibm.com/software/network/hostpublisher, for corrections and additions to this information.

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.

For more information

To access the online Host Publisher Administrator's and User's Guide installed with Host Publisher, use a Web browser to open the following HTML file on your local system:

AIX	/usr/lpp/HostPublisher/common/doc/*lang*/guide/guide.htm
NetWare	//*ServerName*/_IBM_HP_doc_/guide/guide.htm where *ServerName* is the name of your server. **Note:** If *ServerName* is not the same as the IP Host Name, use the IP Host Name in the URL.
OS/400	/QIBM/ProdData/HostPublisher/doc/guide/guide.htm
Solaris	/opt/HostPublisher/common/doc/*lang*/guide/guide.htm
S/390	/usr/lpp/HostPublisher/common/doc/*lang*/guide/guide.htm
Windows NT	*install_dir*\Common\doc\guide\guide.htm

install_dir is the directory in which Host Publisher is installed.

lang is the language-specific subdirectory for your language, and is one of the following:

de_DE	German
en_US	English
es_ES	Spanish
fr_FR	French
it_IT	Italian

ja_JP	Japanese
ko_KO	Korean
pt_BR	Brazilian Portuguese
tr_TR	Turkish
zh_CN	Simplified Chinese
zh_TW	Traditional Chinese

Online help, including the HTML version of this book, is available from the product's user interface.

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

You can use the Integration Objects that Host Publisher creates with integrated development environments (IDEs), such as IBM VisualAge for Java and Symantec Visual Café for Java. You can also use Integration Objects with servlets. This chapter provides instructions for setting up an IDE to work with an Integration Object, instructions for writing an application or servlet, and sample code to use when you set up the runtime environment.

Note: Since the Integration Objects will be running outside of the WebSphere Application Server environment, chaining of Integration Objects is not supported.

Preparing an Integration Object to run in an IDE or with a servlet

These instructions assume you have already created an Integration Object using the Host Publisher **Database Access** or **Host Access** application, and that the IDE is running or the servlet is built on the machine where the Host Publisher Server is installed.

Your integrated development environment (IDE) or servlet will need access to your Integration Object's JAR file. This file is located in the `*Studio_Install_Dir*\Studio\IntegrationObjects\` directory, where *Studio_Install_Dir* is the Host Publisher installation directory for the Host Publisher Studio. Copy this file to the machine where the Host Publisher Server is installed.

To provide the server access to the Integration Object's connection pool specifications, copy the following files from *Studio_Install_Dir*`\Studio\SessionDefs\` to *Server_Install_Dir*`\Server\production\poolspecs\`, where *Server_Install_Dir* is the Host Publisher Server installation directory:

- *connection_name*.connspec
- *connection_name*.poolspec
- *connection_name*.userpool

where *connection_name* is the name of the connection you are using for this Integration Object.

This connection can be the connection you specified when you created your Integration Object or a different connection. See "Writing an application to invoke an Integration Object" on page 3 or "Writing a servlet to invoke an Integration Object" on page 5 for details on how to use a connection other than the one you specified when you created the Integration Object. If you created your Integration Object using the **Database Access** application and selected **prompt for connection values at runtime**, you will not have a *connection_name*.userpool file for this connection.

If you created your Integration Object using the **Host Access** application, also copy the following files from *Studio_Install_Dir*`\Studio\SessionDefs\` to *Server_Install_Dir*`\Server\production\poolspecs\`, where *Server_Install_Dir* is the Host Publisher Server installation directory:

- *connection_name*logon.macro
- *connection_name*logoff.macro
- *connection_name*logspec.logonspec

where *connection_name* is the name of the connection you are using for this Integration Object.

Your IDE or servlet will need access to the following common files located in the `Server_Install_Dir\Common` directory:
- HpRte.jar
- ras.jar
- HPubCommon.jar
- habeansnlv.jar

Copy the following common files, located in the `Studio_Install_Dir\Studio` directory, to the machine where the Host Publisher Server is installed:
- xml4j_ws.jar
- jsdk.jar

Setting up VisualAge for Java to run an Integration Object

You must have installed IBM VisualAge for Java Version 2.0 with Service Pack IV2-2 (or later) to run Integration Objects. (Service Pack IV2-2 is available on the VisualAge Web page as ROLLUP2.)

To set up VisualAge:
1. Import the Common files listed in "Preparing an Integration Object to run in an IDE or with a servlet" on page 1, making sure you include all resource files as well as class files.
2. If you created your Integration Object using **Database Access** application and your database driver is not the JDBC-ODBC Bridge, import the JAR or ZIP file for the database driver.
3. Import the JAR file for the Integration Object.

You can use VisualAge's Type Browser and select the BeanInfo tab to view the Integration Object's properties and methods that are available for use. These methods are also described in "Writing an application to invoke an Integration Object" on page 3.

You are now ready to begin writing.

Setting up Symantec Visual Café to run an Integration Object

To set up Visual Café:
1. Update your SC.INI classpath setting to include the Common files listed in "Preparing an Integration Object to run in an IDE or with a servlet" on page 1.
2. If you created your Integration Object using **Database Access** application and your database driver is not the JDBC-ODBC Bridge, update your SC.INI classpath to include the JAR or ZIP file for the database driver.
3. Select **Add Component to Library** and add the JAR file of the Integration Object. Select **Project**, **Options...**, and **Directories**, and add the JAR file to the **Input Class Files** list. When you add your Integration Object to the Library, you will probably see several error messages that say an "exception occurred

while trying to get the initial value of a property". Ignore these messages. They are generated by Visual Café for the getter methods of the Integration Object that have initial values of null.

If you click on the Integration Object that you added to the Library, its properties are available in the Property List window of Visual Café. However, not all properties of the Integration Object will appear in the Visual Café Property List. Visual Café might not show properties that are indexed (arrays, for example) or do not have both a getter and setter.

You are now ready to begin writing.

Writing an application to invoke an Integration Object

To write an application that invokes an Integration Object:

1. Initialize and start the server.
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()
   ```
   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 *Host Publisher Administrator's and User's Guide*. 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. If you are using VisualAge for Java, use VisualAge's Type Browser to view the BeanInfo.

Writing a servlet to invoke an Integration Object

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 an Integration Object to run in an IDE or with a servlet" on page 1.

To write a servlet that invokes an Integration Object:

1. Create an instance of your Integration Object by calling its constructor.
2. 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 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 you want to use.
3. You must invoke the following method to either specify an error page or specify null if you don't want to redirect to an error page. To specify the error page, invoke the method

   ```
   void setHPubErrorPage(String)
   ```

   specifying the name of your error page relative to the location of your servlet. For example, if your servlet is located one directory above your error page, specify your error page as "../ErrorPage.jsp"). To specify that you do not want to redirect to an error page, invoke this method with a null string.

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. If you redirected to an error page, you should check for error messages by getting the exception object set by your Integration Object using your HttpSession object. Refer to the Host Publisher default error page, which can be found in your `Server_Install_Dir\Server\production\documents` directory, for a sample.

   If you did not redirect to an error page, invoke the methods described in "Writing an application to invoke an Integration Object" on page 3 to get the exception objects.

   Note: If you redirected to an error page, you must add code to your error page to invalidate the HttpSession object if one was created for you. Refer to the Host Publisher default error page, which can be found in your `Server_Install_Dir\Server\production\documents` directory, for a sample of this code. Add this code after you have finished using the HttpSession object.
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. If you are using VisualAge for Java, use VisualAge's Type Browser to view the BeanInfo.

Sample code for initializing and starting the runtime

```
Hashtable initParam = new Hashtable();
initParam.put(com.ibm.HostPublisher.Server.ServerConstants.ADMIN_PARAM_INSTALL_DIR,
     new String("Server_Install_Dir"));
try {
     com.ibm.HostPublisher.Server.Runtime.init(initParam);
     com.ibm.HostPublisher.Server.Runtime.start();
    }
catch (com.ibm.HostPublisher.Server.RteException e )
    {
     // catch and process exception
    }
catch (com.ibm.HostPublisher.Server.RteNeedPassword e )
    {
     // catch and process exception
    }
catch (com.ibm.HostPublisher.Server.RteIsRunning e )
    {
     // catch and process exception
    }
catch (com.ibm.HostPublisher.Server.RteNotInitialized e )
    {
     // catch and process exception
    }
```

where *Server_Install_Dir* specifies the Host Publisher Server installation directory.

The exceptions in the sample code are:

RteException
A fatal error occurred (such as an invalid `ADMIN_PARAM_INSTALL_DIR`)

RteNeedPassword
A startup password is required because of encrypted user pools.

RteIsRunning
The system is already started.

RteNotInitialized
The system has not been initialized.

You need to determine how to process each exception and add your own code for the processing.

Chapter 2. Using the XML Legacy Gateway

The Host Publisher XML Legacy Gateway communicates with a host application in XML data format, and optionally in HTML format. The XML data interface is encapsulated using a JavaBean. The XML data representing the host application screens can be modified and integrated into other Java applications, applets, JavaBeans, or servlets. Interaction with the host program can be manipulated by Java using XML formatting and techniques. See Figure 1 for an architectural overview of the XML Legacy Gateway feature of Host Publisher.

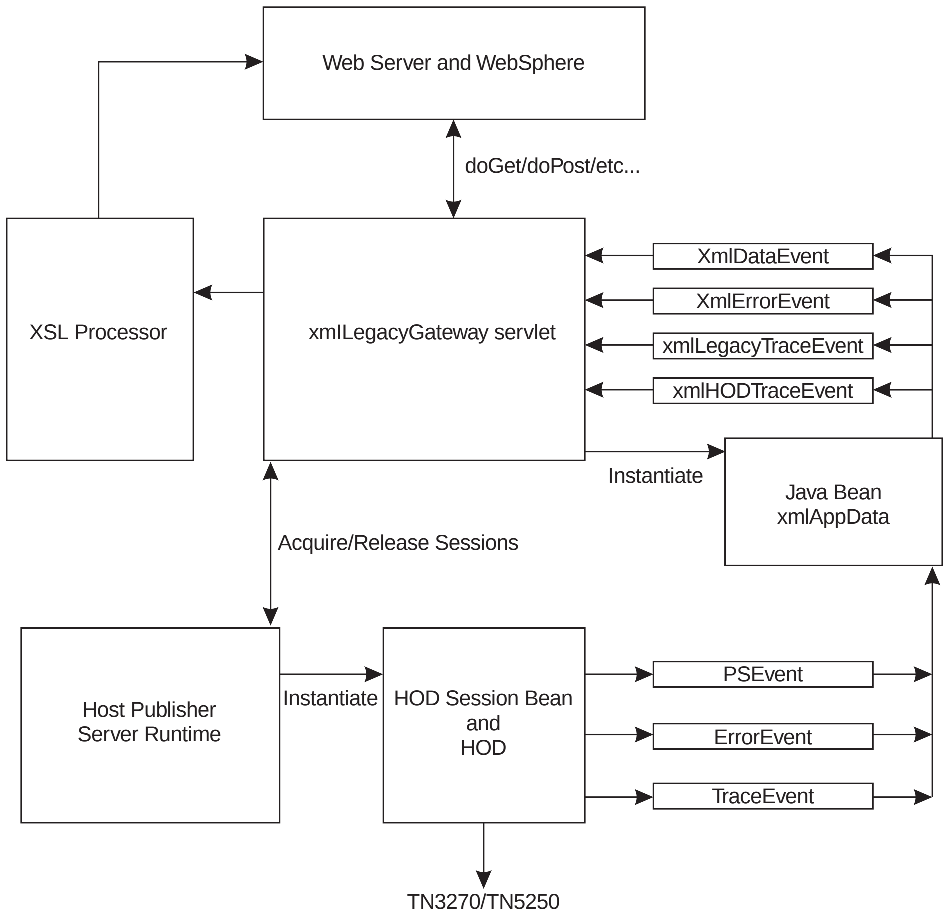

Figure 1. XML Legacy Gateway Architecture

The XML Legacy Gateway provides two components that communicate with a host application:

- The xmlLegacyGateway servlet
- The xmlAppData JavaBean

The xmlLegacyGateway servlet

The xmlLegacyGateway 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.

The 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 this 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. The source code for the servlet is included with the Host Publisher product.

Refer to Figure 1 to see how the xmlLegacyGateway servlet relates to the other components of the XML Legacy Gateway.

The xmlLegacyGateway 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 JavaBean and makes the JavaBean 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 JavaBean.
5. Formats the XML data as HTML output using an XSL stylesheet and an XSL processor. 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 sytlesheet when processing the data. The 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 servlet can do this at run time by specifying the XSL stylesheet used for this particular instance of the servlet.

To facilitate the writing of Host Publisher XML Legacy Gateway servlets, the source code for the xmlLegacyGateway 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\Sample.html
```

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

The xmlAppData JavaBean

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

```
install_dir\SDK\XLGW\xmlAppDataBean\AllNames.html
```

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

Refer to Figure 1 to see how the xmlAppData JavaBean relates to the other components of the XML Legacy Gateway.

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

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

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

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

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

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

The HostConnection JavaBean

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

```
install_dir\SDK\XLGW\HostConnectionBean\AllName.html
```

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

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

Chapter 3. Using Remote Integration Objects

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

You can customize the lightweight Java program for specific business needs, such as correlating data with other JavaBeans 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

The following steps create a Java program (applet or application) to access Integration Object data on a remote machine. In these steps, *prefix* is the value you specify in the **Remote Integration Object Properties...** dialog on the Host Access or Database Access Options menu (the default is Remote), and *IOName* is the name you gave to the Integration Object.

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 RIO files. Refer to "Remote Integration Object files" 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 to perform whatever task is required to access the Integration Object data.
5. Compile the sample CustomApp*IOName*.java file, which is located in the `\HostPub\Studio\IntegrationObjects\RemoteIO\`*`IOName`*`\` directory, with the Java compile command:

   ```
   javac -classpath \HostPub\Common\nano.zip;\HostPub\Studio\RIO.jar;
                    \HostPub\Common\HPubCommon.jar;\HostPub\Common;
                    %classpath% CustomAppIOName.java
   ```
6. For a Java application, transfer the sample CustomApp*IOName*.class program, the RIO supporting JAR files (nano.zip, RIO.jar, and HPubCommon.jar), and the RIO proxy file (IntegrationObject\*prefixIOName*.class) to the remote machine to execute the java application. JDK 1.1.7 or higher is required for the remote machine.

   For a Java applet, transfer the CustomApp*IOName*.class program, AppLoader*IOName*.html file, and RIO supporting JAR files (nano.zip, RIO.jar, and HPubCommon.jar) to the desired HTTP server for browser download.

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 for an Integration Object named TestDB in the `\HostPub\Studio\IntegrationObjects\RemoteIO\TestDB\` directory.

IntegrationObject\RemoteTestDB.java	RIO proxy source file
IntegrationObject\RemoteTestDB.class	RIO proxy class file
CustomAppTestDB.java	Sample Java program source that calls RIO 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

Obtaining Integration Object data in XML format

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 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/servlet/RIOServlet?hPubIntegrationObjectName=IntegrationObject.TestDB
        &hPubRequestType=requestInputs
```

To execute an Integration Object with optional style sheet processing:

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

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

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

Where: TestDB is the name of the Integration Object to execute
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. 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 `\HostPub\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)>
]>
```

Chapter 4. Host Publisher File formats

Host Publisher produces applications using standard open formats—such as HTML pages, Java class files, 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 the 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 the 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 the 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. Other files are only used in the Host Publisher Studio. The sections below describe each file, explain how it is used, and provide the file format.

Integration Object project (.hpi) file

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.

```
<?xml version="1.0" standalone="yes"?>
<!DOCTYPE com.ibm.HostPublisher.IntegrationObject SYSTEM "io.dtd" []>
<com.ibm.HostPublisher.IntegrationObject name = "callup"   type="hod">
     <Package name = "IntegrationObject"/>
      <Session>
      <PoolName>callup</PoolName>
      </Session>

      <OutputVariable name="callupResults" type ="simple">
           <ScreenCoordinates x="2" y="14" dx="71" dy="6"/>
                <SubVariable name="column1" type ="array">
                    <RelativeCoordinates x="0" y="0" dx="26" dy="5"/>
               </SubVariable>
               <SubVariable name="column2" type ="array">
                    <RelativeCoordinates x="26" y="0" dx="4" dy="5"/>
               </SubVariable>
               <SubVariable name="column3" type ="array">
                    <RelativeCoordinates x="30" y="0" dx="9" dy="5"/>
               </SubVariable>
               <SubVariable name="column4" type ="array">
                    <RelativeCoordinates x="39" y="0" dx="9" dy="5"/>
               </SubVariable>
               <SubVariable name="column5" type ="array">
                    <RelativeCoordinates x="48" y="0" dx="9" dy="5"/>
               <SubVariable name="column6" type ="array">
                    <RelativeCoordinates x="57" y="0" dx="15" dy="5"/>
               </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"?>
<!DOCTYPE com.ibm.HostPublisher.IntegrationObject SYSTEM "io.dtd" []>
<com.ibm.HostPublisher.IntegrationObject name = "QuerySample"   type="db">
     <Package name = "IntegrationObject"/>
      <Session>
      <PoolName>callup</PoolName>
      </Session>

      <SQL>SELECT "SHERRI"."DEPARTMENT"."DEPTNO", "SHERRI"."DEPARTMENT"."DEPTNAME",
                     "SHERRI"."DEPARTMENT"."MGRNO" FROM "SHERRI"."DEPARTMENT"
                     WHERE ( ( "SHERRI"."DEPARTMENT"."DEPTNO" <> 'CHY' )
                     AND ( "SHERRI"."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**.

HODMacro

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

JDBCDriver

name The JDBC driver name to use for the specified URL.

JDBCUrl

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

OutputVariable

Output variables define data that will be extracted during the macro execution.

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 using the **REPEAT** tag on a JSP page.

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.

Session

PoolName
Specifies the connection pool name used by this Integration Object.

SessionChain

StartState name
The connection start state label given by the user.

EndState name
The connection end state label given by the user.

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

SQL Identifies the SQL statement to execute for database 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 using the **REPEAT** tag on a JSP page.

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

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'?>
<!DOCTYPE application SYSTEM 'WebBridge.dtd'>
<application>
<appl_name>testdb</appl_name>
<integration_object>
     <obj_name>D:\HostPublisher\Studio\IntegrationObjects\EmployeeQuery.jar
     </obj_name>
     <input_properties>
          <input>setLastName</input>
     </input_properties>
     <output_properties>
          <output>getEmployeesEmployeeIDResult</output>
          <output>getEmployeesFirstNameResult</output>
          <output>getEmployeesExtensionResult</output>
          <output>getEmployeesLastNameResult</output>
          <output>getLastName</output>
     </output_properties>
     <execution_method>doHPTransaction</execution_method>
</integration_object>
<page>d:\hostpublisher2\Studio\testdb\output.jsp</page>
<page>d:\hostpublisher2\Studio\testdb\input.jsp</page>
</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.

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 JavaBean, this is typically the setter method for a JavaBean 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 JavaBean 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 an Java object. For a JavaBean, this is typically the getter method for a JavaBean 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 Objects created by Host Publisher Studio are JavaBeans. The JavaBean files are contained within a JAR file and are generally made up of two files, the JavaBean class and the JavaBean BeanInfo class. These class files are generated based on a template maintained by the Host Publisher Studio and information provided by you through one of the Host Publisher Access applications.

Since the template is not available to you for customization, and since any time the Integration Object is modified using a Host Publisher 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.

JavaServer Pages (JSP) Web page files

Host Publisher Studio generates JSP 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.

The following are sample JSP pages, followed by a description of how the tags are being used.

JSP page for the EmployeeQuery Integration Object

```
<HTML>
<BEAN NAME="EmployeeQuery" TYPE="IntegrationObject.EmployeeQuery"
     INTROSPECT="yes" CREATE="yes" SCOPE="request">
<BEAN>
<% EmployeeQuery.setHPubStartPoolName("NorthwindLocal"); %>
<% EmployeeQuery.doHPTransaction(request, response); %>
<BODY>
<P>Table:
<TABLE BORDER>
<th>EmployeesEmployeeID</th>
<th>EmployeesFirstName</th>
<th>EmployeesExtension</th>
<REPEAT INDEX=idx1>
<tr>
<td>
<INSERT BEAN =EmployeeQuery PROPERTY  =EmployeesEmployeeIDResult>
<INSERT>
</td>
<td>
<INSERT BEAN =EmployeeQuery PROPERTY  =EmployeesFirstNameResult>
</INSERT>
</td>
<td>
<INSERT BEAN =EmployeeQuery PROPERTY  =EmployeesExtensionResult>
</INSERT>
</td>
</REPEAT>
</TABLE>
</BODY>
</HTML>
```

JSP page for the QuerySample Integration Object

```
<HTML>
<BODY>
<BEAN NAME="QuerySample" TYPE="IntegrationObject.QuerySample"
     INTROSPECT="yes" CREATE="yes" SCOPE="request">
<BEAN>
<% QuerySample.setHPubStartPoolName("samplePool"); %>
<% QuerySample.doHPTransaction(request, response); %>
<P>FORM METHOD="POST" ACTION="<%= response.encodeUrl("querySample.jsp") %>">
<P>P>Department Number
<SELECT NAME ="SHERRIDEPARTMENTDEPTNO_" MULTIPLE SIZE=3>
<REPEAT INDEX=idx1>
<% String str = "<OPTION VALUE=\"" +QuerySample.getSHERRIDEPARTMENTDEPTNO_(idx1)
                    + "\">\n"; out.println(str); %>
<INSERT BEAN =QuerySample PROPERTY = SHERRIDEPARTMENTDEPTNO_>
</INSERT>
</OPTION
</REPEAT>
</SELECT>
<P>Department Name <INPUT TYPE = "text" NAME = "Deptname"
                        VALUE ="<%=QuerySample.getDeptname() %>">
<P>INPUT TYPE="submit" VALUE="Submit">
</FORM>
</BODY>
<% out.close(); %>
</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 JSP tags are used on these pages:

BEAN The BEAN tag identifies a Java object to be instantiated on the page. It contains the following parameters:

CREATE
Specifies whether a new instance of this class is to be created. Valid values are **yes** and **no**. Examples of when you would set this value to **no** are:

- The Java class can be manipulated statically without the need for an instance of that class.
- Another JSP page created this Java object instance with a SCOPE value of **session** (see the definition of the **SCOPE** attribute).

INTROSPECT
This turns on automatic introspection for this Java object. This allows a Web parameter, such as one passed into the page via another input form page, to automatically have its value set as an input to the Java object if the parameter has the same name as an input property. Valid values are yes and no. For example, if the Web parameter were **name=Pam** and the EmployeeQuery Integration Object had a property called name, then WebSphere Application Server would automatically set the name property of EmployeeQuery to **Pam**.

NAME
This is the identifier that is used to reference the instantiated Java object on the page. In the first example, this is the EmployeeQuery object. In the second example, this is the QuerySample object. For convenience, it has the same value as the base name of the Integration Object's full class-name (IntegrationObject.EmployeeQuery or IntegrationObject.QuerySample), but it does not have to.

SCOPE
This attribute specifies how long the instance of the Java object is to last. Valid values are **request** and **session**.

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 JSP pages, for example, to access this object again.

TYPE This refers to the Java object's full class-package name.

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
This 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, this 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.

INDEX
This parameter specifies to the **REPEAT** tag an optional index variable that can be used within inline Java code to access the current index value of the repeat loop.

Inline Java tag (<% %>)
These 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
This tag specifies a variety of editable fields inside a form. It contains the following parameters:

NAME
This parameter specifies the variable name for the **VALUE** parameter.

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

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

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

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

image The **INPUT** element is an active inline image.

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

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

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

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

VALUE
This parameter specifies the initial value of the **INPUT** tag.

INSERT

This tag inserts an output property value from an Integration Object or other Java object at the specified location. For the table in the example, the insert position is a cell within a row of the table. Notice how the **REPEAT** tags enclose the creation of a row of data for the table. The **REPEAT** will continue until there are no more rows of data to extract. The **INSERT** tag can only be used on JavaBeans.

Java objects that are not JavaBeans require inline Java code to access the output method using the index variable on the **REPEAT** tag. For example, if there was a Java object on the page called myObject with an indexed output method called getMyData(), then an inline Java statement might look like:

```
<= myObject.getMyData(idx1); %>
```

The equal sign after the inline Java tag (<%=) specifies that the return value from the function should be displayed. Remember, in order for the REPEAT to be terminated, getMyData(idx1) must throw an ArrayIndexOutOfBounds exception when there is no more data to extract, or the REPEAT will continue indefinitely.

OPTION

This 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.

REPEAT

The **REPEAT** and end REPEAT (**</REPEAT>**) tags denote a section of the page that is to be executed repeatedly until WebSphere Application Server receives an ArrayIndexOutOfBounds exception. The only way to get this exception is through the use of indexed properties using the **INSERT** JSP tag or by throwing the exception yourself using the inline Java code tags. Therefore, **REPEAT** tags should not be used without JSP tags between them.

SELECT

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

MULTIPLE

This 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

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, there is no checkin screen for the connection pool.

Application manifest (.application)
: This file describes all Web pages, Integration Objects and other Java objects, and configuration files that an application requires.

  This file is generated by Host Publisher Studio when transferring an application to Host Publisher Server. When the application is deployed into production by the server, this file ensures that all parts of an application are moved into the production area.

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 37 for more information on the format of these XML files.

Note: 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.

Format of connection pool specification files

XML tag conventions

The following sections describe the XML syntax used to define Host Publisher connections and 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 (in the `production/poolspecs` directory) 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.

    Note: The file can be in a subdirectory relative to the `c:/HostPublisher/Server/production/poolspecs` directory, where *c:/HostPublisher* is the directory where Host Publisher is installed. As in all other cases, relative pathnames are specified using the separator character '/'.
- 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.

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 pool is defined for this connection, the user ID/password pairs in that user pool 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 pool, 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 pool is defined for this connection, the first user ID/password pair in the user pool 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>
<connecttimeout value="120"/>
  <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, after which a connection is reclaimed, given the assumption that the Integration Object that acquired the connection is never going to release it.

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.

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 Administration 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 ended and discarded.

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

- **maxidletime**
- **maxbusytime**
- **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">
 <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
: This value is only applicable if connection pooling is enabled. This is a string representation of a com.ibm.eNetwork.ECL.ECLScreenDesc object that is constructed in the Host Publisher Studio. 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 if connection pooling is enabled.

logoffmacro
: References a file containing the Host On-Demand disconnect macro (in Host On-Demand 4.0-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 4.0-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"/>
</hodlogonspec>
</logonconfig>
```

XML Tags for user pool specifications

A user pool is 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 pool 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 pool 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 pool 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 pool is strongly encrypted. The value for the `session` attribute is set when a user of Host Publisher Studio transfers a user pool to the server and selects strong encryption. The user is prompted for a password to be used for strong encryption. When a Web application containing a strongly encrypted user pool is deployed on the Server, the password the user specified for strong encryption must be specified when the Server is restarted. Without this password, the Server cannot be restarted. This password is also required by Host Publisher Server Administration when an administrator modifies the strongly encrypted user pool. If another user pool 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 pool. The Host Publisher Server can only have one startup password, so the same password must be used by all strongly encrypted user pools that are to be deployed to the same server. 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>
<userconfig>
<schema>
   <defineproperty encrypt="0" name="_userid"/>
   <defineproperty encrypt="2" name="_password"/>
</schema>
<localuserpool name="empdbusers"
session="AAEU2JptbII7I3UDqtdMHyu590xGde+p+oU8nxtjdMEisfJp0CsSOMd4gUdpbU5y7kmRe">
   <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>
```

The application manifest file

This file describes, using XML, all the resources that a Host Publisher Studio-generated application depends on. The application manifest lists the pool configuration files, Integration Objects and other Java objects, and JSP and HTML pages used by the application. The manifest is used by Host Publisher Server during deployment to move files from the `staging` directory to the `production` directory and is also used to remove applications and provide administrative information.

All document parts that are private to an application, such as Web pages and images, are copied to an application-specific subdirectory that mirrors the structure in the Web pages. The Host Publisher Studio is not able to determine the location of objects added by a developer if they are not stored using relative paths. For example, if the developer adds HTML to include images using a URL that specifies an HTTP:// URL, or the files are stored with absolute paths that do not exist on the development machine, the Host Publisher Studio ignores these files when transferring files to the Host Publisher Server. If the Host Publisher Studio can locate the file, it is listed in the application manifest file as part of the list of files to be transferred to the server.

There is one manifest file per application. Its name is derived from the application name and should match the subdirectory name under the directory where application-specific Web document files are transferred; for example, the *`c:/HostPublisher`*`/Server/staging/applications` subdirectory, where *c:/HostPublisher* is the directory where Host Publisher is installed. The file has the extension .application, and it is located in the application-specific subdirectory. During deployment the manifest file is moved to the *`c:/HostPublisher`*`/Server/production/appmanifests` subdirectory.

XML Tags for the application manifest

There are three main categories of tags in this file, which are used to reference the parts of an application:

- The **<hodpoolspec>** and **<dbpoolspec>** tags are used to identify one or more connection pools used by the application. These tags identify a connection pool specification file that has been placed in the `staging/shared` directory during the transfer step. The connection pool specification file is used to identify all other files that together define the pool characteristics.
- The **<bean>** and **<beandir>** tags are used to reference the Java objects (for example, Integration Objects) that are used by the application. Either a specific class or JAR file, or an entire directory of class files, can be specified. All directory names are relative to the `staging/shared` directory where Java files are transferred.
- The **<document>** and **<documentdir>** tags are used to reference all the Web document parts (HTML, JSP, GIF, JPEG, and other files) that are used by the application. Either a specific file, or an entire directory of files, can be specified. All directory names are relative to the application-specific subdirectory of the `staging/applications` directory where the application's Web document files are placed during the transfer step.

  Note: All Web document files in the application-specific subdirectory referenced in the manifest are moved to the *`c:/HostPublisher`*`/Server/production/documents` subdirectory during deployment, where *c:/HostPublisher* is the directory where Host Publisher is installed. During the file copy, the application-specific subdirectory is also copied. The URLs used to access the Web pages must include the application name.

The following example illustrates the contents of the application manifest for a (host-based) application called callup. This file is called callup.application, and will be in the *`c:/HostPublisher`*`/Server/staging/applications/callup` directory, where *c:/HostPublisher* is the directory where Host Publisher is installed, after the transfer to the server has been completed.

```
<?xml version="1.0"?>
<!DOCTYPE applconfig SYSTEM "applconfig.dtd">
<applconfig>
<hodpoolspec refname="callup">
<bean filename="com/xyzcorp/callup/callup1.class"/>
<bean filename="com/xyzcorp/callup/callup2.class"/>
<bean filename="xyzcorpmisc.jar"/>
<beandir name="com/xyzcorp/utils"/>
<!-- .java files too, for debugging on the server ? -->
<document filename="callup1.jsp/>
<document filename="callup2.jsp"/>
<document filename="startpage.html"/>
<document filename="errors/errorpage.html"/>
<documentdir name="images"/>
<!-- Entire directory contents should be published/deployed -->
</applconfig>
```

Directory contents before deployment

The following are the contents of the `/var/HostPublisher/Server/staging` directory (for AIX) after the callup application is transferred to the server.

- The `shared` subdirectory contains the following directories/files:
  - com/xyzcorp/callup/callup1.class
  - com/xyzcorp/callup/callup2.class

  - xyzcorpmisc.jar
  - com/xyzcorp/utils/util1.class
  - com/xyzcorp/utils/util2.class
  - Configuration files that together define the connection poolspec for the callup application.
    - callup.poolspec
    - vm6conn.connspec
    - vm6.logonspec
    - vm6_logon.macro
    - vm6_logoff.macro
    - vm6users.userpool
    - vm6_checkin.screen
- The `applications/callup` directory contains the following directories/files:
  - callup1.jsp, callup2.jsp
  - startpage.html
  - errors/errorpage.html
  - images/logo.gif, images/pic1.jpg, images/pic2.jpg
  - callup.application

Directory contents after deployment

The following are the contents of the `/var/HostPublisher/Server/production` directory (for AIX) after Host Publisher Server Administration has copied the files from the `staging` directory during the deployment step.

- The `beans` subdirectory contains the following directories/files:
  - com/xyzcorp/callup/callup1.class
  - com/xyzcorp/callup/callup2.class
  - xyzcorpmisc.jar
  - com/xyzcorp/utils/util1.class
  - com/xyzcorp/utils/util2.class
- The `documents/callup` subdirectory contains the following directories/files:
  - callup1.jsp, callup2.jsp
  - startpage.html
  - errors/errorpage.html
  - images/logo.gif, images/pic1.jpg, images/pic2.jpg
- The `documents/poolspecs` directory contains the following files:
  - callup.poolspec
  - vm6conn.connspec
  - vm6.logonspec
  - vm6_logon.macro
  - vm6_logoff.macro
  - vm6users.userpool
  - vm6_checkin.screen
- The `appmanifests` directory contains the file callup.application.

Macro script files

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

Macro scripts can also be edited directly on the Server. The connect and disconnect macros are stored in the *`Server_Install_Dir`*`\Server\production\poolspecs` directory, where *Server_Install_Dir* is the Host Publisher Server installation directory, and data macros are stored in the *`Server_Install_Dir`*`\Server\production\beans` directory.

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

Macro editing tips

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.

Macro script syntax

This section is excerpted from the *Host On-Demand Bean Reference*. The rest of this book is available on the Web, at http://www.ibm.com/software/network/hostondemand/library under "Host Access Beans for Java".

Introduction

The IBM Host On-Demand V4 Macro and MacroManager beans now use 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, you do this all over again.

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 that you understand that a macro is a series of screens with their actions associated with them, 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>
<screen name="Logon" startscreen="true">
     <description>
     <string value="Please Log on" casesense="true"/>
     <cursor row="12" col="10"/>
     </description>
     <actions>
         <prompt name="ID" row="12" col="10" len="8"/>
         <prompt name="Password" row="13" col="10" len="8"/>
         <input value="[enter]"/>
     </actions>
     <nextscreens>
          <nextscreen name="Logon.Complete"/>
     </nextscreens>
</screen>
<HAScript>
```

These few lines of code demonstrate the power of this new 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.

With the above example, keep in mind that the actions happen in sequence. The **<screen>** tag describes a logon screen with the text **Please Log on** on the screen and the screen's cursor position at row 12, 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, 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 element:

```
<HAScript>
<screen>
<comment>
<description>
<oia>
```

```
<cursor>
<numfields>
<numinputfields>
<string>
<attrib>
<customreco>
<actions>
<prompt>
<input>
<extract>
<message>
<trace>
<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 .2 seconds. The value must be a number.

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> element. The default is one minute. The value must be a number.

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

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.

startscreen
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 startscreen attribute set to true.

stopscreen
If true, a match on the screen causes the macro to stop playing. You can have multiple screens with the stopscreen 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.

Example:

```
<screen name="screen1" startscreen="true" stopscreen="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.

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

Example:

```
<description> ... </description>
```

<numfields> tag: The **<numfields>** tag defines the total number of fields on the screen. This element 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 tag.

optional
: If true, recognition matching passes the element, if they and all other description elements that are not optional match; or, if there are no non-optional elements, and at least one optional element matches. The value must be true or false. This element is optional. The default is false.

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. These element is optional. The default is false.

Example:

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

<numinputfields> tag: The **<numinputfields>** tag defines the total number of input fields on the screen. This element 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 tag.

optional
: If true, recognition matching passes the element, if they and all other description elements that are not optional match; or, if there are no non-optional elements, and at least one optional element matches. The value must be true or false. This element is optional. The default is false.

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. These element is optional. The default is false.

Example:

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

<oia> tag: The **<oia>** tag specifies an operator information area (OIA) condition to match. This element 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 tag.

optional
: If true, recognition matching passes the element, if they and all other

description elements that are not optional match; or, if there are no non-optional elements, and at least one optional element matches. The value must be true or false. This element is optional. The default is false.

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. These element is optional. The default is false.

Example:

```
<oia status="NOTINHIBITED" 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 tag.

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 element is optional.

erow The ending row position for string in a rectangle. The value must be a number. This element 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 element 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 element is optional. The default is false.

optional
If true, recognition matching passes the element, if they and all other description elements that are not optional match; or, if there are no non-optional elements, and at least one optional element matches. The value must be true or false. This element is optional. The default is false.

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. These element is optional. The default is false.

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" />
```

<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 tag.

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

optional
If true, recognition matching passes the element, if they and all other description elements that are not optional match; or, if there are no non-optional elements, and at least one optional element matches. The value must be true or false. This element is optional. The default is false.

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. These element is optional. The default is false.

Example:

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

<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 tag.

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

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

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

optional
If true, recognition matching passes the element, if they and all other description elements that are not optional match; or, if there are no non-optional elements, and at least one optional element matches. The value must be true or false. This element is optional. The default is false.

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. These element is optional. The default is false.

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 tag.

Example:

```
<customreco id="id1" />
```

<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 tag.

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

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

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

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

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

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

encrypted
Use a password echo character. The value must be true or false. This element 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 Access 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"/>
```

<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 element is optional.

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

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

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

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

Example:

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

<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 element is optional. This defaults to current cursor position.

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

movecursor
The place the cursor at the end of the input string. The value must be true or false. This element 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 tag.

xlatehostkeys
If true, host key mnemonics (example, [enter]) will be translated. The value must be true or false. This element is optional. The default is true.

Example:

```
<input value="Your[tab]message[enter]" row="1" col="1" movecursor="true"
     xlatehostkeys="true">
```

<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 element 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 tag.

Example:

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

<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 IBM Host On-Demand's trace facility, a user trace event, or to the command line. Respectively, the types are HODTRACE, USER, SYSOUT. This is a required tag.

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 tag.

Example:

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

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

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

Example:

```
<pause value="10000" />
```

<custom> tag: The **<custom>** tag enables the user to have an exit to Java code, see Host On-Demand's 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 tag.

args The argument string that can be passed to the callout. This can be any valid Unicode character. This element 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. No text is allowed for the tag.

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 element 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 tag.

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. No text is allowed for the tag.

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 tag.

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. This element is optional. If no goto screen is given, the macro terminates.

Example:

```
<recolimit value="3" goto="endscreen"/>
```

A powerful macro

This section gets its name because the following example demonstrates how all of the tags and their attributes can be used in a macro.

```
<HAScript name="Logon Macro" description="Logs me on" author="btwebb"
     creationdate="12/29/1998" promptall="false" pausetime="500" timeout="10000">
     <screen name="Logon" startscreen="true">
     <comment>
     The screen description and actions for this screen demonstrate
     how to use everything. Normally, a screen tag would not be so full.
    </comment>
<description>
     <oia status="NOTINHIBITED" optional="false" invertmatch="false" />
     <string value="Please Log on" casesensitive="true"/>
     <cursor row="12" col="10"/>
     <numinputfields number="2" optional="false" invertmatch="false" />
     <numfields number="10" optional="false" invertmatch="false" />
     <string value="Welcome" row="1" col="1" optional="false"
          invertmatch="false"/>
     <string value="Enter ID" row="1" col="1" erow="11" ecol="11" casesense="false" optional="false"
          invertmatch="false" />
     <string value="USERID" />
     <attrib value="0x01" row="1" col="1" plane="COLOR_PLANE" optional="false" invertmatch="false" />
     <customreco id="logon" />
</description>
<actions promptall="true">
     <prompt name="ID" row="11" col="10" len="8" description="ID for Logon"
          default="btwebb" clearfield="true" encrypted="true" />
     <prompt name="Password" row="13" col="10" len="8"/>
     <extract name="Get Data" srow="1" scol="1" erow="11" ecol="11" />
     <message value="YourMessage" title="Message" />
     <trace value="logging on" type="HODTRACE" />
     <custom id="logon" args="YourArgument" />
     <input value="[enter]" movecursor="true" xlatehostkeys="true"/>
</actions>
<nextscreens timeout="20000" />
     <nextscreen name="Logon.Complete"/>
</nextscreens>
</screen>
<screen name="Logon.Complete" stopscreen="true">
<comment>
     This screen just checks to see if we're logged on OK then hits the ENTER key.
     Because it is a stop screen we don't have to specify any nextscreens tag.
</comment>
<description>
    <oia status="NOTINHIBITED" optional="false" invertmatch="false" />
    <string value="Logon Successful, hit ENTER to continue" casesensitive="true"/>
</description>
<actions>
     <input value="[enter]"/>
</actions>
</screen>
<screen name="MessageReceived" transient="true">
<comment>
     This screen demonstrates the idea of a transient screen. Say our host system can send us
     asynchronous messages while we're logging on, we just want to clear them. This screen
     handles this and is valid anytime a message screen appears. No nextscreens needed here.
</comment>
<description>
     <oia status="NOTINHIBITED" optional="false" invertmatch="false" />
     <string value="Message received, hit CLEAR to continue"/>
</description>
<actions>
     <input value="[clear]"/>
</actions>
</screen>
</HAScript>
```

Recolimit Demonstration

This macro demonstrates how to extract a list that exists on multiple screens using the recolimit and invertmatch attributes. The behavior of this macro is as follows:

1. Assume we get to the ExtractScreen.Main screen (recolimit is 1)
2. Data is extracted and PF8 is sent to page down
3. The ExtractScreen.Main is matched again (recolimit is 2), and so on
4. If recolimit becomes 25, meaning that ExtractScreen.Main was recognized 25 times, the actions for ExtractScreen.Main will not be executed the 25th time. Instead, the actions for ExtractScreen.Complete will be executed. The actions are not executed for ExtractScreen.Main to keep from extracting twice if the ExtractScreen.Complete screen is reached before the recolimit is reached.
5. If ExtractScreen.Complete is matched before the recolimit reaches 25, then we'll just get all the data in the system.

```
<HAScript name="Extracter Macro" description="Gets me data from a list" author="btwebb"
     creationdate="12/29/1998" promptall="false" pausetime="500" timeout="10000">
<screen name="ExtractScreen.Main">
<comment>
     We'll assume there would be other screens that log us on and get us to this point.
     This screen is the main extraction screen, and extracts data only if there is no
     blank line at the bottom of the screen indicating there isn't anymore data
     (invertmatch="true"). To be safe we'll apply a recolimit of 25. This screen does
     an extract, then pages down with PF8.
</comment>
<description>
<oia status="NOTINHIBITED" optional="false" invertmatch="false" />
<string value="Data Screen"/>
<string value="           " row="24" col="1" erow="24" ecol="11" invertmatch="true" />
</description>
<actions promptall="true">
     <extract name="Get Data" srow="2" scol="1" erow="24" ecol="80" />
     <input value="[pf8]"/>
</actions>
```

```
<nextscreens timeout="20000">
    <nextscreen name="ExtractScreen.Main"/>
    <nextscreen name="ExtractScreen.Complete"/>
</nextscreens>
<recolimit value="25" goto="ExtractScreen.Complete"/>
</screen>
<screen name="ExtractScreen.Complete">
<comment>
     This screen is the final extraction screen, and extracts data and sends an exit command.
     Note: It is only different from the main screen in that the blanks must be there.
     Assume there would be other screens to take care of logging off.
</comment>
<description>
     <oia status="NOTINHIBITED" optional="false" invertmatch="false" />
     <string value="Data Screen"/>
     <string value="           " row="24" col="1" erow="24" ecol="11"/>
</description>
<actions promptall="true">
     <extract name="Get Data" srow="2" scol="1" erow="24" ecol="80" />
     <input value="exit[enter]"/>
</actions>
<nextscreens timeout="20000">
     <nextscreen name="ExtractScreen.Complete"/>
</nextscreens>
</screen>
</HAScript
```

The server.properties file

The server.properties file is a Java properties file that contains Host Publisher Server settings. The file is created when Host Publisher Server is installed. The server.properties file is primed with default values when you start the Host Publisher Server for the first time. The server.properties file is written to the `Server` directory of the Host Publisher installation directory on the server. When you use Host Publisher Server Administration to make changes, the values you specify are written to the server.properties file.

If no value is specified for any of the properties listed below, Host Publisher uses the default value defined for that property.

In addition to using Host Publisher Server Administration to update this file, you can edit the server.properties file manually to make changes to the server settings. If you manually edit the file, you should restart the server to ensure that the changes you make take effect.

The server.properties file contains the following properties:

num_licenses
Specifies the number of licenses you purchased.

The value is an integer. The default is 1.

Specify num_licenses = –1 if you purchased an unlimited license.

licenseTracking
Specifies whether Host Publisher tracks license usage or not.

Note: You can only modify this property by editing the server.properties file.

0 Host Publisher does not track license usage.

1 Host Publisher tracks license usage. The Host Publisher Server tracks the number of Host Publisher connections to host or database resources and logs a message when the value exceeds the number of licenses purchased. The license usage information is written to a file named licenseX.txt in the `log` directory of the Host Publisher installation directory on the server, where the X is either 1 or 2.

The maximum size of the license usage files is 512 KB. When the file size of the license1.txt file reaches 512KB or if the Host Publisher Server is restarted, the file is renamed to license2.txt, and a new license1.txt file is created. The new license1.txt file contains the most recent license usage information. When the new license1.txt reaches 512KB and is renamed, the old license2.txt is deleted.

The license usage files contain the following information, arranged in rows, with one row per hour. The values are separated by a comma (,).

1. Date
2. Time
3. The highest license count since the server was started
4. The highest license count in the last hour (the maximum of the last 60 entries)
5. The license count for each minute (1–60)

The value is binary. The default is 0.

%logFile
Specifies the path and file name of the file to which messages are written. A backslash in the path should be specified with a double slash (\\).

maxLogFiles
Specifies the maximum number of files for messages.

Note: You can modify this property only by editing the server.properties file.

In the server.properties file:

1. Set maxLogFiles to the desired number.
2. Optionally, set maxLogFileSize in kilobytes. The default is 512 KB.

The algorithm Host Publisher Server uses to create additional log files when the file in use is full, is as follows: once the log file becomes full for the first time, it is renamed by adding a 1 to the end of the name. For example, messages.txt becomes messages1.txt. As the size of the file grows, the following takes place:

1. Host Publisher writes to messages1.txt until it reaches maxLogFileSize.
2. Host Publisher closes and renames the messages1.txt file to messages2.txt. Host Publisher opens a new file named messages1.txt and starts logging to that file.
3. When messages1.txt is full, Host Publisher renames it to messages2.txt, renames messages2.txt to messages3.txt, and opens a new messages1.txt file and starts logging to that file.
4. When the maxLogFiles number is exceeded, Host Publisher deletes the oldest file, which is the file with the highest number.

There are never more files than the number set for maxLogFiles. For example, if maxLogFiles is 3 and maxLogFileSize is 1000, Host Publisher Server eventually creates three log files named messages1.txt, messages2.txt, and messages3.txt. The most recent log entries appear in messages1.txt, and its size is less than 1000 kilobytes. The older log entries appear in messages2.txt and messages3.txt, and each of these files is approximately 1000 kilobytes in size. All log entries older than those contained in messages3.txt have been discarded.

The value is an integer. The default is 2.

maxLogFileSize
Specifies the maximum size, in kilobytes, that a message log file reaches before an additional log file is opened.

Note: You can only modify this property by editing the server.properties file.

The value is an integer. The default is 512 KB.

%traceFile
Specifies the path and file name of the file to which traces are written. A backslash in the path should be specified with a double slash (\\).

maxTraceFiles
Specifies the maximum number of files for trace information.

Note: You can only modify this property by editing the server.properties file.

In the server.properties file:

1. Set maxTraceFiles to the desired number.
2. Optionally, set maxTraceFileSize in kilobytes. The default is 512.

The algorithm Host Publisher Server uses to create additional trace files when the file in use is full, is as follows: once the trace file becomes full for the first time, it is renamed by adding a 1 to the end of the name. For example, trace.txt becomes trace1.txt. As the size of the file grows, the following takes place:

1. Host Publisher writes to trace1.txt until it reaches maxTraceFileSize.
2. Host Publisher closes and renames the trace1.txt file to trace2.txt. Host Publisher opens a new file named trace1.txt and starts tracing to that file.
3. When trace1.txt is full, Host Publisher renames it to trace2.txt, renames trace2.txt to trace3.txt, and opens a new trace1.txt file and starts tracing to that file.
4. When the maxTraceFiles number is exceeded, Host Publisher deletes the oldest file, which is the file with the highest number. There are never more files than the number set for maxTraceFiles.

The value is an integer. The default is 2.

maxTraceFileSize
Specifies the maximum size, in kilobytes, that a trace file reaches before an additional trace file is opened.

Note: You can only modify this property by editing the server.properties file.

The value is an integer. The default is 512 KB.

%logMask
Specifies the types of messages that are logged. Add the values for each of the following message types to determine the value for this property:

1	Informational messages
2	Warning messages
4	Error messages

The value is a decimal integer. The default is 4.

Note: Error messages are always logged.

%traceMask
Specifies the types of traces for the Server and the Integration Objects. This property does not affect JDBC or Host On-Demand tracing. Add the values for each of the following traces to determine the value for this property:

1	API traces
4	Entry and exit traces
16	Miscellaneous traces

The value is a decimal integer. The default is 0.

%HODDisplayTerminal
Specifies whether Host On-Demand displays a terminal window for each connection or not.

0	Host On-Demand does not display a terminal window for each connection.
1	Host On-Demand displays a terminal window for each connection.

The value is binary. The default is 0.

%HODMacroTracingLevel
Specifies the level of tracing for the Host On-Demand macros.

The value is an integer in the range 0 to 3, where 3 is the maximum level of tracing. The default is 0, which means there is no tracing.

%HODPSTracingLevel
Specifies the level of tracing for the Host On-Demand presentation space.

The value is an integer in the range 0 to 3, where 3 is the maximum level of tracing. The default is 0, which means there is no tracing.

%HODTransportTracingLevel
Specifies the level of tracing for the Host On-Demand transport.

The value is an integer in the range 0 to 3, where 3 is the maximum level of tracing. The default is 0, which means there is no tracing.

%JDBCTracing
Specifies how much JDBC activity Host Publisher traces.

0	Host Publisher does not trace JDBC activity.
1	Host Publisher traces all JDBC activity in the application server.

The value is binary. The default is 0.

%runtimeTracing
: Specifies whether Host Publisher traces runtime activity or not.

0	Host Publisher does not trace runtime activity.
1	Host Publisher does trace the runtime activity.

The value is binary. The default is 0.

%HPAdminFile
: Specifies the path and file name of the file that defines the Host Publisher Server Administration main page. You should never change this value.

autoDeploy
: Specifies whether Host Publisher deploys applications that have been transferred to the server when the server is started.

0	You must deploy published applications manually using Host Publisher Server Administration.
1	Host Publisher deploys applications automatically when the server is started.

The value is binary. The default is 0.

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.

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/390
- VisualAge
- WebSphere

Other company, product, and service names may be trademarks or service marks of others.

ActionMedia, LANDesk, MMX, Pentium and ProShare are trademarks of Intel Corporation in the United States, other countries, or both.

(For a complete list of Intel trademarks see http://www.intel.com/tradmarx.htm)

Adobe is a trademark of Adobe Systems, Incorporated.

C-bus is a trademark of Corollary, Inc. in the United States, other countries, or both.

DIGITAL is a trademark of Digital Equipment Corporation.

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Lotus and Domino are trademarks or registered trademarks of Lotus Development Corporation.

Microsoft, Windows, Windows NT, and FrontPage are trademarks or registered trademarks of Microsoft Corporation in the United States, other countries, or both.

Netscape is a registered trademark of Netscape Communications Corporation in the United States and other countries.

NetWare, Novell Installation Services (NIS), and NetWare Enterprise Web Server are registered trademarks of Novell in the United States and other countries.

Oracle is a registered trademark of Oracle Corporation.

PC Direct is a trademark of Ziff Communications Company in the United States, other countries, or both and is used by IBM Corporation under license.

Sybase and its corresponding logo are property of Sybase, Inc.

UNIX is a registered trademark in the United States and other countries licensed exclusively through The Open Group.

SET and the SET Logo are trademarks owned by SET Secure Electronic Transaction LLC. For further information see http://www.setco.org/aboutmark.html.

Visual Café is a trademark of Symantec Corporation in the United States, other countries, or both.

Index

Special Characters

%HODDisplayTerminal tag 52
%HODMacroTracingLevel tag 52
%HODPSTracingLevel tag 52
%HODTransportTracingLevel tag 52
%HPAdminFile tag 53
%JDBCTracing tag 52
%logFile tag 50
%logMask tag 51
%runtimeTracing tag 52
%traceFile tag 51
%traceMask tag 52
.application, application manifest file 26
.connspec, connection specification file 26
.hpa, Host Publisher application file 20
.hpi, Integration Object project file 17
.java, Integration Object source file 21
.logonspec, logon specification file 26
.macro, macro files 26
.poolspec, connection pool specification file 26
.screen, checkin screen description file 26
.userpool, user pool specification file 26

A

ACTION attribute
  FORM tag 23
actions tag 44
additional information v
application, using
  to invoke an Integration Object 3
application manifest (.application) file 26
application manifest file 34
  example 35
application manifest tags
  bean 35
  beandir 35
  dbpoolspec 35
  document 35
  documentdir 35
  hodpoolspec 35
args attribute
  custom tag 46
attrib tag 43
attributes
  ACTION
    FORM tag 23
  args
    custom tag 46
  author
    HAScript tag 39
  casesense
    string tag 42
  clearfield
    prompt tag 44
  col
    attrib tag 43

attributes *(continued)*
  col *(continued)*
    cursor tag 43
    input tag 45
    prompt tag 44
    string tag 42
  CREATE
    BEAN tag 22
  creationdate
    HAScript tag 39
  default
    prompt tag 44
  description
    HAScript tag 39
    prompt tag 44
  ecol
    extract tag 45
    string tag 42
  encrypted
    prompt tag 44
  EndState name
    SessionChain tag 19
  erow
    extract tag 45
    string tag 42
  filename
    HODMacro tag 18
  goto
    recolimit tag 47
  id
    custom tag 46
  ID
    customreco tag 44
  INTROSPECT
    BEAN tag 23
  invertmatch
    attrib tag 43
    cursor tag 43
    numfields tag 41
    numinputfields tag 41
    oia tag 42
    string tag 42
  len
    prompt tag 44
  METHOD
    FORM tag 23
  movecursor
    input tag 45
  MULTIPLE
    SELECT tag 25
  name
    com.ibm.HostPublisher.IntegrationObject tag 18
    extract tag 45
    HAScript tag 39
    JDBCDriver tag 18
    JDBCUrl tag 18
    nextscreen tag 47
    OutputVariable tag 18
    Package tag 19
    prompt tag 44

attributes *(continued)*
  name *(continued)*
    screen tag 40
    SubVariable tag 19
  NAME
    BEAN tag 23
    INPUT tag 24
    SELECT tag 25
  number
    numfields tag 41
    numinputfields tag 41
  optional
    attrib tag 43
    cursor tag 43
    numfields tag 41
    numinputfields tag 41
    oia tag 41
    string tag 42
  pausetime
    HAScript tag 39
  plane
    attrib tag 43
  PoolName
    Session tag 19
  Position
    SessionChain tag 19
  promptall
    actions tag 44
    HAScript tag 39
  RelativeCoordinates
    SubVariable tag 19
  row
    attrib tag 43
    cursor tag 43
    input tag 45
    prompt tag 44
    string tag 42
  scol
    extract tag 45
  SCOPE
    BEAN tag 23
  ScreenCoordinates
    OutputVariable tag 18
  SIZE
    SELECT tag 25
  srow
    extract tag 45
  startscreen
    screen tag 40
  StartState name
    SessionChain tag 19
  status
    oia tag 41
  stopscreen
    screen tag 40
  timeout
    HAScript tag 39
    nextscreens tag 46
  title
    message tag 45

attributes *(continued)*
transient
screen tag 40
type
com.ibm.HostPublisher.IntegrationObject tag 18
OutputVariable tag 18
SubVariable tag 19
trace tag 46
TYPE
BEAN tag 23
INPUT tag 24
value
attrib tag 43
input tag 45
message tag 45
pause tag 46
recolimit tag 47
string tag 42
trace tag 46
VALUE
INPUT tag 24
xlatehostkeys
input tag 45
prompt tag 44
author attribute
HAScript tag 39
autoDeploy tag 53

B

bean tag 35
beandir tag 35
BeanInfo tab 2

C

casesense attribute
string tag 42
checkin screen description (.screen) file 26
checkinscreendesc tag 32
classpath
SC.INI 2
clearfield attribute
prompt tag 44
col attribute
attrib tag 43
cursor tag 43
input tag 45
prompt tag 44
string tag 42
comment tag 40
common files, Host Publisher Studio
jsdk.jar 2
xml4j_ws.jar 2
common files, Server
habeansnlv.jar 2
HpRte.jar 2
HPubCommon.jar 2
ras.jar 2
complete HOD macro
example 47
connection pool specification
connspec file 1
poolspec file 1
userpool file 1
connection pool specification (.poolspec) file 26
connection specification (.connspec) file 26
connection specification file
example 29
connecttimeout tag 28, 29
connspec file 1
conventions, XML tag 27
CREATE attribute
BEAN tag 22
creationdate attribute
HAScript tag 39
cursor tag 43
custom tag 46
customreco tag 44

D

dbconnspec tag 29
dbpoolspec tag 35
default attribute
prompt tag 44
deployment
file locations after 36
file locations before 35
description attribute
HAScript tag 39
prompt tag 44
description tag 40
disconnecttimeout tag 28
document tag 35
documentation, additional v
documentdir tag 35
drivername tag 28

E

ecol attribute
extract tag 45
string tag 42
editing
files 17
macros manually 37
server.properties file 49
encrypted attribute
prompt tag 44
EndState name attribute
SessionChain tag 19
entry tag 33
erow attribute
extract tag 45
string tag 42
example
application manifest 35
complete HOD macro 47
connection specification file 29
HOD connection macro 32
HOD logon macro 40
pool specification file 31
recolimit 48
user pool definition file 33
extract tag 45

F

FAQs vi
file locations
after deployment 36
before deployment 35
filename attribute
HODMacro tag 18
files
.connspec 1
.poolspec 1
.userpool 1
habeansnlv.jar 2
HpRte.jar 2
HPubCommon.jar 2
jsdk.jar 2
ras.jar 2
SC.INI 2
Server
server.properties 49
Studio
application manifest 34
application manifest (.application) 26
checkin screen description (.screen) 26
connection pool specification (.poolspec) 26
connection specification (.connspec) 26
Host Publisher application (.hpa) 20
Integration Object project (.hpi) 17
Integration Object source (.java) 21
JavaServer Pages (JSP) Web page 21
logon specification (.logonspec) 26
macro (.macro) 26
user pool specification (.userpool) 26
xml4j_ws.jar 2
files, editing 17
files, macro 26
frequently asked questions vi

G

goto attribute
recolimit tag 47

H

habeansnlv.jar file 2
HAScript tag 39
HOD connection macro
example 32
HOD logon macro
example 40
hodconnspec tag 30
hodlogonspec tag 30
hodpoolspec tag 35
Host On-Demand
connection files 28
logon and logoff specification 32
Host On-Demand connection specification tags
connecttimeout 28

Host On-Demand connection specification tags *(continued)*
  disconnecttimeout 28
  sessionprops 28
  singlelogon 28
Host Publisher
  Web site address vi
Host Publisher application (.hpa) file 20
  sample 20
Host Publisher application (.hpa) file tags
  appl_name 20
  execution_method 20
  input 20
  input_properties 20
  integration_object 21
  obj_name 21
  output 21
  output_properties 21
  page 21
Host Publisher Studio, common files
  jsdk.jar 2
  xml4j_ws.jar 2
HostConnection JavaBean 12
HpRte.jar file 2
HPubCommon.jar file 2

I

IBM VisualAge for Java 1
  setting up 2
  Type Browser
    BeanInfo tab 2
id attribute
  custom tag 46
ID attribute
  customreco tag 44
IDE
  IBM VisualAge for Java 1
  Symantec Visual Café 1
idx value 25
information, additional v
input tag 45
integrated development environment (IDE)
  IBM VisualAge for Java 1
  Symantec Visual Café 1
Integration Object
  connection pool specification
    connspec file 1
    poolspec file 1
    userpool file 1
  data in XML format 14
  JAR file 1
  logon specification
    logoff macro 1
    logon macro 1
    logonspec file 1
  remote 13
    creating 13
    files 13
Integration Object methods
  invoking
    with a servlet 5
    with an application 3
Integration Object project (.hpi) file 17
  Database Access sample 18
  Host Access sample 17
Integration Object project file tags
  com.ibm.HostPublisher.IntegrationObject 18
  HODMacro 18
  JDBCDriver 18
  JDBCUrl 18
  OutputVariable 18
  Package 19
  Session 19
  SessionChain 19
  SQL 19
  SubVariable 19
Integration Object source (.java) file 21
Integration Objects
  methods
    viewing 2
  preparing to run 1
  programming with 1
    sample code 7
  properties
    viewing 2
INTROSPECT attribute
  BEAN tag 23
invertmatch attribute
  attrib tag 43
  cursor tag 43
  numfields tag 41
  numinputfields tag 41
  oia tag 42
  string tag 42
invoking an Integration Object
  with a servlet 5
  with an application 3

J

JavaBean
  HostConnection 12
  xmlAppData 11
JavaServer Pages (JSP) Web page file
  samples 21
JavaServer Pages (JSP) Web page file tags
  BEAN 22
  FORM 23
  INDEX 24
  Inline Java (<% %>) 24
  INPUT 24
  INSERT 25
  OPTION 25
  REPEAT 25
  SELECT 25
JavaServer Pages (JSP) Web page files 21
JDBC connection files 28
JDBC connection specification tags
  connecttimeout 28
  drivername 28
  urlname 28
jsdk.jar 2
JSP Web pages 21

L

len attribute
  prompt tag 44
licenseTracking tag 49
localuserpool tag 30, 33
logoff macro 1
logoffmacro tag 32
logon and logoff specification tags
  checkinscreendesc 32
  logoffmacro 32
  logonmacro 32
logon macro 1
logon specification (.logonspec) file 26
logonmacro tag 32
logonspec file 1

M

macro (.macro) files 26
macro script syntax 37
macro syntax tags
  actions 44
  attrib 43
  comment 40
  cursor 43
  custom 46
  customreco 44
  description 40
  extract 45
  HAScript 39
  input 45
  message 45
  nextscreen 47
  nextscreens 46
  numfields 41
  numinputfields 41
  oia 41
  pause 46
  prompt 44
  recolimit 47
  screen 40
  string 42
  trace 46
macros
  editing manually 37
manifest (.application) file, application 26
maxbusytime tag 30
maxconnections tag 30
maxidletime tag 30
maxLogFiles tag 50
maxLogFileSize tag 51
maxTraceFiles tag 51
maxTraceFileSize tag 51
message tag 45
METHOD attribute
  FORM tag 23
methods, Integration Object
  viewing 2
minconnections tag 30
movecursor attribute
  input tag 45
MULTIPLE attribute
  SELECT tag 25

N

name attribute
  com.ibm.HostPublisher.IntegrationObject tag 18
  extract tag 45
  HAScript tag 39
  JDBCDriver tag 18
  JDBCUrl tag 18

name attribute *(continued)*
  nextscreen tag 47
  OutputVariable tag 18
  Package tag 19
  prompt tag 44
  screen tag 40
  SubVariable tag 19
NAME attribute
  BEAN tag 23
  INPUT tag 24
  SELECT tag 25
nextscreen tag 47
nextscreens tag 46
num_licenses tag 49
number attribute
  numfields tag 41
  numinputfields tag 41
numfields tag 41
numinputfields tag 41

O

oia tag 41
optional attribute
  attrib tag 43
  cursor tag 43
  numfields tag 41
  numinputfields tag 41
  oia tag 41
  string tag 42
overflowallowed tag 30

P

pause tag 46
pausetime attribute
  HAScript tag 39
plane attribute
  attrib tag 43
pool specification file
  connection (.poolspec) 26
  example 31
  user (.userpool) 26
pool specification tags
  connecttimeout 29
  dbconnspec 29
  hodconnspec 30
  hodlogonspec 30
  localuserpool 30
  maxbusytime 30
  maxconnections 30
  maxidletime 30
  minconnections 30
  overflowallowed 30
  poolingenabled 30
poolingenabled tag 30
PoolName attribute
  Session tag 19
poolspec file 1
Position attribute
  SessionChain tag 19
programming with Integration Objects 1
  sample code 7
prompt tag 44
promptall attribute
  actions tag 44
  HAScript tag 39
properties, Integration Object
  viewing 2
property tag 33

R

ras.jar file 2
recolimit 48
  example 48
recolimit tag 47
RelativeCoordinates attribute
  SubVariable tag 19
Remote Integration Objects 13
  creating 13
  files 13
row attribute
  attrib tag 43
  cursor tag 43
  input tag 45
  prompt tag 44
  string tag 42

S

sample code
  programming with Integration Objects 7
samples
  Host Publisher application (.hpa) file 20
  Integration Object project (.hpi) file 17
  JavaServer Pages (JSP) Web page file 21
SC.INI file
  classpath 2
schema tag 33
scol attribute
  extract tag 45
SCOPE attribute
  BEAN tag 23
screen description (.screen) file, checkin 26
screen tag 40
ScreenCoordinates attribute
  OutputVariable tag 18
Server, common files
  habeansnlv.jar 2
  HpRte.jar 2
  HPubCommon.jar 2
  ras.jar 2
Server files
  server.properties 49
server properties file, editing 49
server.properties tags
  %HODDisplayTerminal 52
  %HODMacroTracingLevel 52
  %HODPSTracingLevel 52
  %HODTransportTracingLevel 52
  %HPAdminFile 53
  %JDBCTracing 52
  %logFile 50
  %logMask 51
  %runtimeTracing 52
  %traceFile 51
  %traceMask tag 52
  autoDeploy 53
server.properties tags *(continued)*
  licenseTracking 49
  maxLogFiles 50
  maxLogFileSize 51
  maxTraceFiles 51
  maxTraceFileSize 51
  num_licenses 49
servlet
  xmlLegacyGateway 10
servlet, using
  to invoke an Integration Object 5
sessionprops tag 28
singlelogon tag 28
SIZE attribute
  SELECT tag 25
srow attribute
  extract tag 45
startscreen attribute
  screen tag 40
StartState name attribute
  SessionChain tag 19
status attribute
  oia tag 41
stopscreen attribute
  screen tag 40
string tag 42
Studio files
  application manifest 34
  application manifest (.application) 26
  checkin screen description (.screen) 26
  connection pool specification (.poolspec) 26
  connection specification (.connspec) 26
  Host Publisher application (.hpa) 20
  Integration Object project (.hpi) 17
  Integration Object source (.java) 21
  JavaServer Pages (JSP) Web page 21
  logon specification (.logonspec) 26
  macro (.macro) 26
  user pool specification (.userpool) 26
Symantec Visual Café 1
  setting up 2
syntax, macro script 37

T

tag conventions, XML 27
tag descriptions
  application manifest 35
  connection specifications 28
  Host Publisher application (.hpa) 20
  Integration Object project file 18
  JavaServer pages 22
  logon and logoff specification 32
  pool specifications 29
  server properties 49
  user pool specifications 32
timeout attribute
  HAScript tag 39
  nextscreens tag 46
title attribute
  message tag 45
trace tag 46
transient attribute
  screen tag 40

type attribute
  com.ibm.HostPublisher.IntegrationObject
    tag 18
  OutputVariable tag 18
  SubVariable tag 19
  trace tag 46
TYPE attribute
  BEAN tag 23
  INPUT tag 24
Type Browser
  BeanInfo tab 2

U

urlname tag 28
URLs vi
user pool definition file
  example 33
user pool specification (.userpool) file 26
user pool specification tags
  entry 33
  localuserpool 33
  property 33
  schema 33
  userconfig 33
userconfig tag 33
userpool file 1

V

value attribute
  attrib tag 43
  input tag 45
  message tag 45
  pause tag 46
  recolimit tag 47
  string tag 42
  trace tag 46
VALUE attribute
  INPUT tag 24
viewing
  Integration Object methods 2
  Integration Object properties 2

W

Web page vi
Web pages, JSP 21
white papers vi

X

xlatehostkeys attribute
  input tag 45
  prompt tag 44
XML Legacy Gateway 9
  HostConnection JavaBean 12
  servlet 10
  xmlAppData JavaBean 11
XML tag conventions 27
xml4j_ws.jar 2
xmlAppData JavaBean 11
xmlLegacyGateway servlet 10

Readers' Comments — We'd Like to Hear from You

IBM® WebSphere™ Host Publisher
Programmer's Guide and Reference
Version 2 Release 2

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

IBM®

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
Information Development
Department CGMD / Bldg 500
P.O. Box 12195
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.