IBM Host Access Transformation Services

Web Application Programmer's Guide

Version 9.5

SC27-5902-01

IBM Host Access Transformation Services

Web Application Programmer's Guide

Version 9.5

SC27-5902-01

Note

Before using this information and the product it supports, be sure to read the general information under Appendix B, "Notices," on page 165

Tenth Edition (November 2015)

Contents

Figures

Tables

Chapter 1. Introduction

The Host Access Transformation Services (HATS) Toolkit offers many tools for creating and customizing Web HATS applications that provide an easy-to-use graphical user interface (GUI) for your character-based 3270 or 5250 host applications. HATS Web applications, including portlets can be developed with a look and feel that matches your company's Web or portal pages, and your users can access them through their Web browsers. HATS can also be used to create service-oriented architecture (SOA) assets from logic contained in your character based 3270, 5250, or VT applications.

Service-Oriented Architecture expresses a perspective of software architecture that defines the use of loosely coupled software services to support the requirements of the business processes and software users. In an SOA environment, resources on a network are made available as independent services that can be accessed without knowledge of their underlying platform implementation.

SOA can also be regarded as a style of Information Systems architecture that enables the creation of applications that are built by combining loosely coupled and interoperable services. These services operate together based on a formal definition or contract (for example, WSDL) that is independent of the underlying platform and programming language. The interface definition hides the implementation of the language-specific service. SOA-compliant systems can therefore be independent of development technologies and platforms (such as Java™ and .NET). For example, services written in C# running on .NET platforms and services written in Java running on Java EE platforms can both be consumed by a common composite application. In addition, applications running on either platform can consume services running on the other as Web services, which facilitates reuse. SOA can support integration and consolidation activities within complex enterprise systems, but SOA does not specify or provide a methodology or framework for documenting capabilities or services.

You might find that your HATS application requires some additional function that you cannot add using the wizards and editors in HATS Toolkit and IBM® Rational® Software Delivery Platform (SDP). This *Web Application Programmer's Guide* explains several ways that you can extend your HATS application with additional programming. It also assumes that you are familiar with basic HATS concepts such as:

- How HATS processes host screens
- Creating a transformation using components and widgets
- Events and actions
- Using global variables
- Recording a macro
- Creating an Integration Object from a macro

If you are not already familiar with any of these topics, refer to the information about them in *HATS User's and Administrator's Guide* so that you will have the necessary background to make good use of the information in this book. You should also be familiar with using Rational SDP to create Java EE applications.

This *Web Application Programmer's Guide* describes ways to enhance your HATS application by programming. You can:

- Add business logic classes to be invoked as an action when an event occurs. You can also create custom logic to aid in recognizing host screens. See Chapter 2, "Adding business logic," on page 3 for more information.
- Add new host components or widgets to be used in transformations by extending the existing host components and widgets. See Chapter 3, "Creating custom components and widgets," on page 17 for more information.
- Perform several programming tasks with Integration Objects. See Chapter 6, "Programming with Integration Objects," on page 53 and Chapter 9, "Integration Objects - advanced topics," on page 91 for more information.
- Make one or more Integration Objects available as a Web service, which makes the objects available for use by other applications. See Chapter 7, "Developing Web services," on page 65 for more information.
- Make one or more Integration Objects available through an EJB client, which makes them available for use by other applications. See Chapter 8, "Creating and using a HATS EJB application," on page 81 for more information.
- Create your own plug-ins for Web Express Logon. See Chapter 10, "Creating plug-ins for Web Express Logon," on page 103.
- Enhance the capabilities of your HATS portlets. See Chapter 5, "Programming in HATS Portlets," on page 47 for more information.
- Use the HATS bidirectional API to work with the orientation of screen elements in applications that use bidirectional code pages. See Chapter 11, "Using the HATS bidirectional API," on page 111 for more information.

When enhancing your applications, you might find that you need to edit some of the Java source files. Information provided in the section of the Rational Software Delivery Platform help titled *Developing Java applications* can help you with this task.

Code examples

Code examples throughout this guide illustrate the use of the objects or APIs introduced in the adjoining sections. The examples may or may not work if you copy them from the book into your application.

Using the API documentation (Javadoc)

The HATS API reference documentation is useful for many programming tasks. To view this documentation, see IBM Knowledge Center collection for HATS at http://www.ibm.com/support/knowledgecenter/SSXKAY_9.5.0 and click the HATS API References (Javadoc) link. Refer to this documentation when you need information about, and examples of, any of the Application Programming Interfaces provided with HATS.

Chapter 2. Adding business logic

Business logic is any Java code that is invoked as an action when an event occurs, such as a host screen being recognized or your HATS application being started. Business logic is specific to the application and is not provided as part of HATS. You can use business logic to extend your HATS application to integrate with other data sources, such as a database or an Integration Object. For example, you can read the contents of a file or a database into HATS global variables and use the global variables to fill in a drop-down list or pop-up to be used in the application's GUI pages.

HATS automatically updates your class loader policy when you include HATS business logic in your HATS Web project. This updates the default WAR class loader policy of your HATS application to **Single class loader for application**.

You can create business logic and add it to your project using the Create Business Logic wizard. To invoke this wizard, right-click in the **HATS Projects** tab of the HATS Toolkit, and click **New HATS > Business Logic**.

In the Create Business Logic wizard, specify the project you want to add the business logic to and supply the Java class name. The default package name is *projectName*.businessLogic, but you can change this in Studio Preferences. Optionally, you can supply a package name or click **Browse** to select an existing Java package. If you want your business logic to include methods for easy access to the project global variables, or to remove project global variables, select the **Create global variable helper methods** check box. Click **Finish** when you have provided the required information.

After you create your business logic class, you will want to link it to one or more screen or application events so it is executed when that event occurs. Edit each event (application event or screen customization) to which you want to add the link. On the Actions tab, click **Add**, select **Execute business logic**, then fill in the details for your business logic class. Refer to *HATS User's and Administrator's Guide* for information about editing both screen customization and events.

You can see the business logic files in the project by expanding the **Source** folder on the **HATS Project View** tab of the HATS Toolkit. Each package name or class name appears in the **Source** folder. Expand the package name folder to see the Java class name. Click the class name to edit the class. The **Source** folder can also include other Java files that have been imported into your HATS project.

If you use the Create Business Logic wizard to create business logic, the method that is invoked by the execute action is named **execute** by default. If you write your own class, the method must have the following attributes:

- Be marked public and static
- Have a return type of void
- Accept a com.ibm.hats.common.IBusinessLogicInformation object as the only parameter

The method must use this form, followed by your own business logic code:

```
public static void myMethod (IBusinessLogicInformation businesslogic)
```

The IBusinessLogicInformation object that is passed to your custom Java code enables you to access and use or modify various objects and settings of your HATS project. These include:

- The com.ibm.hats.runtime.IRequest class, which returns an object representing the request made to the HATS runtime and provides access to request parameters.
- The com.ibm.hats.runtime.IResponse class, which returns an object representing the response from the HATS runtime.
- The `getConnectionMap()` method, which returns a java.util.Map that contains the settings for the connection information that you provided for the application.
- The `getGlobalVariables()` method, which returns a java.util.Hashtable of global variables for this application instance. This table does not include shared global variables.
- The `getSharedGlobalVariables()` method, which returns a java.util.Hashtable of shared global variables for this application instance.
- Class properties, which provide default settings for objects such as components and widgets
- The com.ibm.hats.common.HostScreen object, which contains host screen information
- The java.util.Locale class of the client
- The com.ibm.hats.common.TextReplacementList values and settings
- The client session identifier string (returned by getter methods in the business logic template that the Create Business Logic wizard provides)
- The current screen orientation of bidirectional sessions
- The existence of the **Screen Reverse** button in the browser for bidirectional sessions

For more information about the classes made available to you, see the HATS API documentation in the HATS Knowledge Center at http://www.ibm.com/support/knowledgecenter/SSXKAY_9.5.0 for the IBusinessLogicInformation class. Since IBusinessLogicInformation extends the IBaseInfo class, several of these APIs are defined in the IBaseInfo class.

Incorporating Java code from other applications

You can incorporate Java code from other existing applications into your HATS projects in a variety of ways.

If you want to incorporate the source code (.java files) from your existing business logic so you can modify the code, you can import the .java files into the **Source** folder in your existing project. Click **File > Import > General > File System** to open the Import wizard. In the Import wizard, select the location of your source files in the **From directory** field. For Web projects, select the **Java Source** folder of your project in the destination **Into folder** entry field. When your source .java files are imported, they are automatically compiled and packaged into your HATS project. You can edit, set breakpoints, and debug your source files in the Rational SDP workbench.

You can also incorporate a Java archive (.jar) file with compiled Java business logic. The entire Java archive is added; you cannot select individual classes to add.

1. Import the .jar file into the HATS EAR project.
   a. Click **File > Import > General > File System** to open the Import wizard.

b. Select the source directory of the Java archive (.jar) you want to import in the **From directory** field.
c. Select the .jar file from the right pane.
d. Select your HATS EAR project as the destination in the **Into folder** field.
e. Click **Finish**.
f. When your .jar file is imported, click the **Navigator** tab of the HATS Toolkit and expand your HATS ear project. You will see the imported java archive file.

2. Add a java archive to the project build path.
   a. In the **Navigator** tab of the HATS Toolkit, select the project in which you want to invoke your business logic.
   b. Right-click the high-level HATS project, and select **Properties**.
   c. In the **Properties** window, select **Java Build Path** in the left table and click the **Libraries** tab on the right.
   d. Click **Add JARs** to open the **JAR Selection** window.
   e. Expand the HATS EAR project, and select the newly imported Java archive file.
   f. Click **OK** in the **JAR Selection** window, and click **OK** in the **Properties** window.
   g. Repeat this process for all HATS projects for which you want to use the business logic.
3. Define the project where the business logic is to be invoked.
   a. In the **Navigator** tab of the HATS Toolkit, again select the project in which you want to invoke your business logic.
   b. Expand the project, the Web Content folder, and the META-INF folder.
   c. Double-click the MANIFEST.MF file to open the JAR dependencies editor. Select the check box next to each .jar file that you want to include in your project's class path.

There are other ways to import Java archives into the HATS project. HATS projects are extensions of Web projects in the Rational SDP workbench. For more information about importing files into Web projects, open the Rational SDP Help and search for **Web projects**

Using global variables in business logic

If your HATS application uses global variables to store information, you can use these global variables in your business logic.

There are two types of global variables: local and shared. A local global variable is one that is created within a HATS project and is only visible to the project. A shared global variable is one that is visible to and can be used by all the applications in an EAR file. There are also two lists of HATS global variables, one for local global variables and one for shared global variables. Two global variables with the same name can coexist if one is local and the other is shared.

When you create your business logic class, use the Create Business Logic wizard and select the **Create global variable helper methods** check box. This creates methods in your business logic for getting, setting, and removing local and shared global variables.

The following methods are created:

```
////////////////////////////////////////////////////////////////////////////
//  This sample is provided AS IS.
//  Permission to use, copy and modify this software for any purpose and
//  without fee is hereby granted. provided that the name of IBM not be used in
//  advertising or publicity pertaining to distribution of the software without
//  specific written permission.
//
//  IBM DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SAMPLE, INCLUDING ALL
//  IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL IBM
//  BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY
//  DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER
//  IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING
//  OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SAMPLE.
////////////////////////////////////////////////////////////////////////////
/**
 * Example method that sets a named global variable
 * from the current session to a value
 * @param blInfo - IBusinessLogicInformation from current session
 * @param name - Name of the global variable
 * @param value - Value of the global variable
 */
public static void setGlobalVariable(IBusinessLogicInformation blInfo,
     String name,Object value)
{
  IGlobalVariable gv = blInfo.getGlobalVariable(name);
  if ( gv == null )
   {
    gv = new GlobalVariable(name,value);
   }
  else
   {
    gv.set(value);
   }
  blInfo.getGlobalVariables().put(name,gv);
}

/**
 * Example method that sets a named shared
 * global variable from the current session to a value
 * @param blInfo - IBusinessLogicInformation from current session
 * @param name - Name of the shared global variable
 * @param value - Value of the shared global variable
 */
public static void setSharedGlobalVariable(IBusinessLogicInformation
  blInfo,String name,Object value)
{
  IGlobalVariable gv = blInfo.getSharedGlobalVariable(name);
  if ( gv == null )
   {
    gv = new GlobalVariable(name,value);
   }
  else
   {
    gv.set(value);
   }
  blInfo.getSharedGlobalVariables().put(name,gv);
}

/**
 * Example method that removes a named global variable from the current session
 * @param blInfo - IBusinessLogicInformation from current session
 * @param name - Name of the global variable
*/
public static void removeGlobalVariable(IBusinessLogicInformation blInfo, String  name)
{
  IGlobalVariable gv = blInfo.getGlobalVariable(name);
  if ( gv != null )
   {
    blInfo.getGlobalVariables().remove(name);
    gv.clear();
    gv = null;
   }
}
/**
 * Example method that removes a named shared global variable from the current session
 * @param blInfo - IBusinessLogicInformation from current session
 * @param name - Name of the shared global variable
*/
public static void removeSharedGlobalVariable(IBusinessLogicInformation blInfo, String name)
{
```

```
  IGlobalVariable gv = blInfo.getSharedGlobalVariable(name);
   if ( gv != null )
    {
     blInfo.getSharedGlobalVariables().remove(name);
     gv.clear();
     gv = null;
    }
}
/**
 * Example method that retrieves a named global variable from the current session
 * @param blInfo - IBusinessLogicInformation from current session
 * @param name - Name of the global variable
 * @return - an instance of the global variable, or null if not found.
 */
public static IGlobalVariable getGlobalVariable(IBusinessLogicInformation
  blInfo,String name)
{
  IGlobalVariable gv = blInfo.getGlobalVariable(name);
  return gv;
}

/**
 * Example method that retrieves a named shared
 * global variable from the current session
 * @param blInfo - IBusinessLogicInformation from current session
 * @param name - Name of the shared global variable
 * @return - an instance of the global variable, or null if not found.
 */
public static IGlobalVariable getSharedGlobalVariable(IBusinessLogicInformation
  blInfo,String name)
{
  IGlobalVariable gv = blInfo.getSharedGlobalVariable(name);
  return gv;
}
```

Elsewhere in your code, when you need the value of a local global variable, you can call this method:

```
GlobalVariable gv1 = getGlobalVariable(blInfo,"varname");
```

To get the value of a shared global variable, use the following method:

```
GlobalVariable gv1 = getSharedGlobalVariable(blInfo,"varname");
```

Business logic examples

This section contains examples of using business logic. Each works with global variables. Each example uses one or more of the global variable helper methods previously described, and the classes should include those methods. They are omitted in these examples to make it easier to view the example code.

Example: Date conversion

This example converts a date from mm/dd/yy format to month, day, year format. For example, the example converts 6/12/2004 into June 12, 2004. The example assumes that the global variable *theDate* has been set before the business logic is called. Note how the example uses the following method to obtain the value of the input variable:

```
IGlobalVariable inputDate = getGlobalVariable(blInfo, "theDate");
```

After using standard Java functions to manipulate the string to represent the date in the desired format, the example uses the following method to put the new string into the same global variable:

```
setGlobalVariable(blInfo, "theDate", formattedDate);
////////////////////////////////////////////////////////////////////////////
//  This sample is provided AS IS.
//  Permission to use, copy and modify this software for any purpose and
```

```
//  without fee is hereby granted. provided that the name of IBM not be used in
//  advertising or publicity pertaining to distribution of the software without
//  specific written permission.
//
//  IBM DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SAMPLE, INCLUDING ALL
//  IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL IBM
//  BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY
//  DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER
//  IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING
//  OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SAMPLE.
////////////////////////////////////////////////////////////////////////////
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Date;

import com.ibm.hats.common.IBusinessLogicInformation;
import com.ibm.hats.common.GlobalVariable;
import com.ibm.hats.common.IGlobalVariable;

public class CustomDateFormatter
{
   public static void execute(IBusinessLogicInformation blInfo)
   {
    IGlobalVariable inputDate = getGlobalVariable(blInfo, "theDate");
    SimpleDateFormat inputFormat = new SimpleDateFormat("MM/dd/yyyy");
    SimpleDateFormat outputFormat = new SimpleDateFormat("MMMM dd, yyyy");

    try
     {
      Date tempDate = inputFormat.parse(inputDate.getString().trim());
      String formattedDate = outputFormat.format(tempDate);
      setGlobalVariable(blInfo, "theDate", formattedDate);
     }
      catch (ParseException ex)
       {
        ex.printStackTrace();
       }
   }
}
```

Example: Adding values that are contained in an indexed global variable

This example adds the values that are contained in an indexed global variable and stores the sum in another, non-indexed global variable. It assumes that you have stored strings representing numbers in the indexed global variable *subtotals*.

The previous example included the names of the input and output global variables (*theDate*) on the set calls. This example sets the names of the input and output variables into local string variables and uses those strings on calls to get and set the global variable values. Because the name of the global variable is being passed as a variable, it is not put into quotes:

```
setGlobalVariable(blInfo,gvOutputName, new Float(myTotal));
```

```
////////////////////////////////////////////////////////////////////////////
//  This sample is provided AS IS.
//  Permission to use, copy and modify this software for any purpose and
//  without fee is hereby granted. provided that the name of IBM not be used in
//  advertising or publicity pertaining to distribution of the software without
//  specific written permission.
//
//  IBM DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SAMPLE, INCLUDING ALL
//  IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL IBM
//  BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY
//  DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER
//  IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING
//  OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SAMPLE.
////////////////////////////////////////////////////////////////////////////
import com.ibm.hats.common.IBusinessLogicInformation;
import com.ibm.hats.common.GlobalVariable;
import com.ibm.hats.common.IGlobalVariable;

  public static void execute(IBusinessLogicInformation blInfo)
```

```
{
 // Name of indexed global variable to be read in
    String gvInputName = "subtotals";
    // Name of global variable to be calculated and saved
    String gvOutputName = "total";

    // The indexed global variable where each index is a subtotal to be summed
    GlobalVariable gvSubtotals =
 ((GlobalVariable)getGlobalVariable(blInfo, gvInputName));

    float myTotal = 0;

    // Calculate the total by adding all subtotals
    for (int i = 0; i < gvSubtotals.size(); i++)
    {
     myTotal = myTotal + Float.valueOf(gvSubtotals.getString(i)).floatValue();
    }

    // Save the total as a non-indexed local variable
    setGlobalVariable(blInfo,gvOutputName, new Float(myTotal));
}
```

Example: Reading a list of strings from a file into an indexed global variable

This example reads a file from the file system and stores the strings in the file into an indexed global variable. You can use a technique like this to read a file that contains, for example, a list of your company's locations. After storing the strings in a global variable, you can use the global variable to populate a drop-down list or other widget to enable users to select from a list of values. You can create a global rule to use this widget wherever an appropriate input field occurs. To make sure that the global variable is available as soon as the application is started, add the execute action for this business logic class to the Start event.

Note: If your text file has carriage returns and line feeds between lines, you might need to use "\r\n" as the second argument of the StringTokenizer constructor call in the following example.

```
////////////////////////////////////////////////////////////////////////////
//  This sample is provided AS IS.
//  Permission to use, copy and modify this software for any purpose and
//  without fee is hereby granted. provided that the name of IBM not be used in
//  advertising or publicity pertaining to distribution of the software without
//  specific written permission.
//
//  IBM DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SAMPLE, INCLUDING ALL
//  IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL IBM
//  BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY
//  DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER
//  IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING
//  OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SAMPLE.
////////////////////////////////////////////////////////////////////////////
import com.ibm.ejs.container.util.ByteArray;
import com.ibm.hats.common.IBusinessLogicInformation;
import com.ibm.hats.common.GlobalVariable;
import com.ibm.hats.common.IGlobalVariable;

public class ReadNamesFromFile
{

  public static void execute(IBusinessLogicInformation blInfo)
  {
  // Name of indexed global variable to be saved
  String gvOutputName = "namesFromFile";

  // The file containing a list of information (in this case, it contains names)
  java.io.File myFileOfNames = new java.io.File("C:" + java.io.File.separator
       + "temp" + java.io.File.separator + "names.txt");

     try
     {
   // First, read the contents of the file

   java.io.FileInputStream fis = new java.io.FileInputStream(myFileOfNames);
```

```
   int buffersize = (int)myFileOfNames.length();
   byte[] contents = new byte[buffersize];;

   long n = fis.read(contents, 0, buffersize);

   fis.close();

   String namesFromFile = new String(contents);

   // Next, create an indexed global variable from the file contents

   java.util.StringTokenizer stok =
      new java.util.StringTokenizer(namesFromFile, "\n", false);

   int count = stok.countTokens();
   String[] names = new String[count];
   for (int i = 0; i < count; i++)
   {
    names[i] = stok.nextToken();
   }

   IGlobalVariable gv = new GlobalVariable(gvOutputName, names);
   blInfo.getGlobalVariables().put(gvOutputName,gv);
     }
     catch (java.io.FileNotFoundException fnfe)
     {
      fnfe.printStackTrace();
     }
     catch (java.io.IOException ioe)
     {
      ioe.printStackTrace();
     }
  }
}
```

Example: Calling an Integration Object

This example illustrates calling an Integration Object from business logic. The example assumes that you have an Integration Object named Iomac, which takes one input parameter, a string called input, and returns a string named output.

This business logic performs these steps:

1. Instantiates the Integration Object:

   ```
   IntegrationObject.Iomac io = new IntegrationObject.Iomac();
   ```

2. Sets the input variable from a global variable:

   ```
   io.setInput(getGlobalVariable(blInfo,"input").getString());
   ```

3. Gets the servlet request and response objects from the HATS wrapper method and invokes the Integration Object using the request and response:

   ```
   WebRequest webReq = (WebRequest)blInfo.getRequest();
   HttpServletRequest req = webReq.getHttpServletRequest();
   WebResponse webResp = (WebResponse)blInfo.getResponse();
   HttpServletResponse resp = webResp.getHttpServletResponse();
   io.doHPTransaction(req,resp);
   ```

4. Checks for exceptions.

5. If the Integration Object executed successfully, sets the global variable output to the value returned by the Integration Object's `getOutput()` method:

   ```
   if (io.getHPubErrorOccurred() == 0)
      setGlobalVariable(blInfo,"output",io.getOutput());
   ```

If you want to invoke an Integration Object that accepts more input variables or returns more variables, add setter and getter calls to set the input variables before invoking the Integration Object and retrieve the output values after it executes.

```
////////////////////////////////////////////////////////////////////////////
//  This sample is provided AS IS.
//  Permission to use, copy and modify this software for any purpose and
//  without fee is hereby granted. provided that the name of IBM not be used in
```

```
//  advertising or publicity pertaining to distribution of the software without
//  specific written permission.
//
//  IBM DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SAMPLE, INCLUDING ALL
//  IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL IBM
//  BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY
//  DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER
//  IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING
//  OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SAMPLE.
////////////////////////////////////////////////////////////////////////////
import com.ibm.hats.common.IBusinessLogicInformation;
import com.ibm.hats.common.GlobalVariable;
import com.ibm.hats.common.IGlobalVariable;

public class myLogic
{

    public static void execute(IBusinessLogicInformation blInfo)
    {
        //add code here to perform your business logic
        IntegrationObject.Iomac io = new IntegrationObject.Iomac();

  // Set Integration Object's HAOVariable "input"
  // equal to text from HATS gv "input"
  io.setInput(getGlobalVariable(blInfo,"input").getString());

  // Get the request and response objects from the HATS wrapper method and run the hp transaction

  try {
   WebRequest webReq = (WebRequest)blInfo.getRequest();
   HttpServletRequest req = webReq.getHttpServletRequest();
   WebResponse webResp = (WebResponse)blInfo.getResponse();
   HttpServletResponse resp = webResp.getHttpServletResponse();
   io.doHPTransaction(req,resp);
  }
  catch (Exception e) {
   System.out.println("Exception thrown: " + e.toString());
   setGlobalVariable(blInfo,"output",
    "An exception was thrown. Please view the log for more details");
  }

  // Retrieve "output" computed through IO transaction
  // and set it to HATS gv "output"

  if (io.getHPubErrorOccurred() == 0) {
   setGlobalVariable(blInfo,"output",io.getOutput());
   System.out.println("Transaction has been completed successfully.");
  }
  else {
   setGlobalVariable(blInfo,"output","Transaction has failed unexpectedly.
     HATS Error message = " + io.getHPubErrorMessage());
   System.out.println("Transaction failed unexpectedly. HATS Error message =
     " + io.getHPubErrorMessage()");  }
    }
}
```

Note: To run an Integration Object from business logic in standard portlets, the processRequest() method must be called instead of doHPTransaction(HttpServletRequest, HttpServletResponse). You can use the following code in step 3 (above) instead:

```
io.setHPubStartPoolName("my_hats_project/default_connection");
io.processRequest();
```

If the Integration Object is configured to use Web Express Logon in standard portlets, you must ensure PortletRequest is available in the Integration Object by calling setHPubPortletRequest(javax.portlet.PortletRequest) before executing the processRequest() method. For example:

```
io.setHPubStartPoolName("my_hats_project/default_connection");
  com.ibm.hats.portlet.runtime.JsrStandardPortletRequest jsrReq =
 (com.ibm.hats.portlet.runtime.JsrStandardPortletRequest)blInfo.getRequest();
  javax.portlet.PortletRequest req = jsrReq.getPortletRequest();
  io.setHPubPortletRequest(req);
  io.processRequest();
```

Using custom screen recognition

You can use business logic to perform custom screen recognition. HATS Toolkit provides many options for recognizing screens within a screen customization, including the number of fields on a screen, the presence or absence of strings, and the use of global variables. These options are described in *HATS User's and Administrator's Guide*. You might find, however, that you want to recognize a screen in a way that you cannot configure using the options in the screen customization editor. In that case, you can add your own custom screen recognition logic.

Note: The information in this section can be used for screen recognition within macros as well as within screen customizations.

If you want to create custom screen recognition logic using HATS global variables, see "Custom screen recognition using global variables" on page 14.

If you have already created custom screen recognition logic by extending the ECLCustomRecoListener class, you can use this logic within HATS. If you are creating new custom logic, follow these steps:

1. Open the Java perspective.
2. Click **File > New > Class**.
3. Browse to the Source directory of your HATS project.
4. Enter the names of your package and class.
5. For the superclass, click **Browse** and locate com.ibm.hats.common.customlogic.AbstractCustomScreenRecoListener.
6. Select the check box for **Inherited abstract methods**. Click **Finish**. This imports the code skeleton into the project you specified.
7. Add your logic to the `isRecognized` method. Make sure that it returns a boolean value.

   ```
   public boolean isRecognized(String arg0, IBusinessLogicInformation
   arg1, ECLPS arg2, ECLScreenDesc arg3)
   ```

   Refer to the HATS API documentation at the HATS Knowledge Center for a description of this method.
8. After creating your method, you must update the screen recognition to invoke your method. From the HATS Projects view, expand your project and the **Screen Customizations** folder. Double-click the name of the screen customization to which you want to add your custom logic. Click the **Source** tab to open the Source view of the screen customization.
9. Within the Source view, you will see a block that begins and ends with the <description> and </description> tags. This block contains the information that is used to recognize screens. Add a line within this block to invoke your custom logic:

   ```
   <customreco id="customer.class.package.MyReco::settings"
   invertmatch="false" optional="false"/>
   ```

   where customer.class.package.MyReco is your package and class name. If you want to pass any settings into your class, add them after the class name, separated by two colons. Settings are optional, and your class must parse whatever values are passed in. If you do not need settings, omit the two colons.

   Consider where within the description block you want to place the <customreco> tag. If you want your custom logic invoked only if all the other criteria match, place the <customreco> tag at the end of the block,

immediately before the </description> tag. If your screen customization compares a screen region to a value, the description block will contain a smaller block, beginning and ending with the <block> and </block> tags, to define the value to which the screen region is compared. Be sure not to place your customreco tag inside this block.

Following is an example section of a description block. Note the <customreco> tag just before the </description> tag, and not between the <block and </block> tags.

```
<description>
<oia invertmatch="false" optional="false" status="NOTINHIBITED"/>
<numfields invertmatch="false" number="61" optional="false"/>
<numinputfields invertmatch="false" number="16" optional="false"/>
<block casesense="false" col="2" ecol="14" erow="21"
   invertmatch="false" optional="false" row="20">
<string value="USERID ==="/>
<string value="PASSWORD ==="/>
</block>
<cursor col="16" invertmatch="false" optional="false" row="20"/>
<customreco id="customer.class.package.MyReco::settings"
   invertmatch="false" optional="false"/>
</description>
```

10. To rebuild your HATS project, click **Project > Clean** on the toolbar.
11. Use Run on Server to test your project. If you receive a ClassNotFoundException error, modify the class loader policy on your server. Refer to *HATS Getting Started* for more information.

Example of custom screen recognition

Following is an example of business logic that performs custom screen recognition. This business logic class takes a list of code page numbers, separated by blanks, as its settings, and recognizes the screen if its code page matches one of those listed in the settings. The tag syntax is:

```
<customreco id="company.project.customlogic.CodePageValidate::[settings]"
optional="false" invertmatch="false" />
```

For example, you can insert the following tag into a description block:

```
<customreco id="company.project.customlogic.CodePageValidate::037 434 1138"
optional="false" invertmatch="false" />
```

In this case the screen will be recognized if its code page is 037, 434, or 1138.

```
////////////////////////////////////////////////////////////////////////////
//  This sample is provided AS IS.
//  Permission to use, copy and modify this software for any purpose and
//  without fee is hereby granted. provided that the name of IBM not be used in
//  advertising or publicity pertaining to distribution of the software without
//  specific written permission.
//
//  IBM DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SAMPLE, INCLUDING ALL
//  IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL IBM
//  BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY
//  DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER
//  IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING
//  OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SAMPLE.
////////////////////////////////////////////////////////////////////////////
package company.project.customlogic;

import java.util.StringTokenizer;
import java.lang.Integer;
import com.ibm.eNetwork.ECL.ECLPS;
import com.ibm.eNetwork.ECL.ECLScreenDesc;
import com.ibm.hats.common.IBusinessLogicInformation;
import com.ibm.hats.common.HostScreen;
import com.ibm.hats.common.customlogic.AbstractCustomScreenRecoListener;

public class CodePageValidate extends AbstractCustomScreenRecoListener {
```

```
/**
 * @see com.ibm.hats.common.customlogic.AbstractCustomScreenRecoListener
 * #isRecognized(java.lang.String, com.ibm.hats.common.IBusinessLogicInformation,
 *       com.ibm.eNetwork.ECL.ECLPS,  com.ibm.eNetwork.ECL.ECLScreenDesc)
  */
 public boolean isRecognized(
  String settings,
  IBusinessLogicInformation bli,
  ECLPS ps,
  ECLScreenDesc screenDescription) {
  HostScreen hs=bli.getHostScreen();
  int int_codepage=hs.GetCodePage();
  if(settings!=null)
  {
   StringTokenizer tokenizer = new StringTokenizer(settings);
   while( tokenizer.hasMoreTokens() )
   {
    int int_token= Integer.valueOf(tokenizer.nextToken()).intValue();
    if ( int_token==int_codepage )
    {
      return true;
    }
   }
  }
  return false;
 }

}
```

Custom screen recognition using global variables

HATS Toolkit provides some screen recognition options using global variables, including these functions:

- Verify that a global variable exists
- Verify that a global variable does not exist
- Verify the integer or string value of a global variable

Refer to *HATS User's and Administrator's Guide* for information about these options. If you want to perform screen recognition that is based on HATS global variables and the options in the Global Variable Logic panel do not meet your requirements, you can add your own logic based on the values or existence of one or more global variables. This approach does not require you to create a Java class; instead, it uses the GlobalVariableScreenReco class, which is provided by HATS, and you can specify comparisons to be made as settings on the <customreco> tag. The format is one of the following:

- ```
  <customreco
  id="com.ibm.hats.common.customlogic.GlobalVariableScreenReco::
  {variable(name,option,resource, index)}COMPARE{type(name,option,resource,
                                                      index)}"
    invertmatch="false" optional="false"/>
  ```
- ```
  <customreco
  id="com.ibm.hats.common.customlogic.GlobalVariableScreenReco::
  {variable(name,option,resource, index)}COMPARE{type(value)}"
  invertmatch="false" optional="false"/>
  ```

Braces {} are used to contain each of the two items that are being compared. The first item is a HATS global variable, whose name is specified in *name*. You can use `option` to specify that you want to use the variable's value, length, or existence in your comparison. The `resource` and `index` settings are optional. Use `resource` to indicate whether the global variable is local (which is the default) or shared. Use `index` to indicate which value to use from an indexed global variable.

The second item can be one of the following:

- Another HATS global variable, with similar options, in which case the first format is used
- A fixed value, in which case the second format is used

The valid values for the settings are shown in Table 1. For the COMPARE setting, the only valid values for comparing strings are EQUAL and NOTEQUAL.

Table 1. Valid values for settings

Setting	Valid values
type	• variable • integer • boolean • string
COMPARE	• EQUAL • NOTEQUAL • GREATERTHAN • GREATERTHANOREQUAL • LESS THAN • LESSTHANOREQUAL
options	• exists (boolean) • value (string/integer/boolean) • length (integer) • object (object)
resource	• local • shared
index	Any positive integer or 0

The following example compares the values of two local global variables:

```
<customreco id="com.ibm.hats.common.customlogic.GlobalVariableScreenReco::
  {variable(name=gv1,option=value,resource=local)}EQUAL
  {variable(name=gv2,option=value,resource=local)}"
  invertmatch="false" optional="false"/>
```

This expression evaluates to true if the values of *gv1* and *gv2* are the same.

Now consider the length option. For a non-indexed global variable, length is the length of the value of the variable. For an indexed global variable, if you specify an index, length is the length of that index of the global variable; if you do not specify an index, length is the number of indexed entries in the global variable.

```
<customreco id="com.ibm.hats.common.customlogic.GlobalVariableScreenReco::
  {variable(name=gv1,option=length,resource=shared)}LESSTHANOREQUAL
  {variable(name=gv2,option=length,index=4)}" invertmatch="false" optional="false"/>
```

This expression compares the length of *gv1* to the length of the fourth index of *gv2*. It evaluates to true if the length of *gv1* is less than or equal to the length of the fourth index of *gv2*. You can use LESSTHANOREQUAL because length returns an integer value.

The use of resource=shared on *gv1* in this example indicates that *gv1* is a shared global variable. The other option is resource=local, which is the default, and means that the global variable is not shared with other applications.

You do not have to compare one global variable with another. You can compare the length of a global variable with a fixed integer. You can compare the value of a global variable with another string. For example:

```
<customreco id="com.ibm.hats.common.customlogic.GlobalVariableScreenReco::
  {variable(name=gv1,option=value)}EQUAL{string(value=mystring)}"
  invertmatch="false" optional="false"/>
```

This expression compares the value of *gv1* with the string that is contained in mystring. The string can be a fixed string, the value of a variable, or a value that is returned from a method call. In general, you do not need to use custom logic to compare the length or value of a global variable to a fixed value; you can add these comparisons using the Global Variable Logic panel.

Accessing javax.servlet classes

In order to provide a consistent set of APIs for both Web and rich client application developers, some HATS APIs were generalized so they would work in either a Web or rich client environment. For example, prior to HATS 7.0, the javax.servlet.HttpServletRequest object for a request could be accessed via the BusinessLogicInfo.getRequest() method. Although this API is still available (but deprecated), new business logic classes use the new IBusinessLogicInformation interface instead of the BusinessLogicInfo class.

The getRequest() method on this interface returns a com.ibm.hats.runtime.IRequest object. In a Web application, this object will be of type com.ibm.hats.runtime.WebRequest. WebRequest has similar methods to HttpServletRequest, but does not extend from it. To access the actual HttpServletRequest object from a WebRequest object, the getHttpServletRequest() method can be called.

Table 2. Javax.servlet class plus HATS class plus the method to access

javax.servlet Class	+ Wrapper HATS Class	+ Method to access
HttpServletRequest	com.ibm.hats.runtime.WebRequest	WebRequest.getHttpServletRequest()
HttpServletResponse	com.ibm.hats.runtime.WebResponse	WebResponse.getHttpServletResponse()
ServletContext	com.ibm.hats.runtime.WebContext	WebContext.getServletContext()
ServletConfig	com.ibm.hats.runtime.WebConfig	WebConfig.getServletConfig()

See the Javadoc API for IBusinessLogicInformation for more information and code samples.

Chapter 3. Creating custom components and widgets

HATS provides a set of host components that recognize elements of the host screen and widgets that render the recognized elements. The components and widgets have settings that you can modify if the default settings do not recognize components or render widgets as you want them. If the components, widgets, and the settings that are provided by HATS do not meet your needs, you can create your own custom components or widgets or modify existing host components or widgets. You might want to create your own host component in order to recognize elements of your host screen that the HATS components do not recognize. You might want to create your own widget in order to change the way elements are presented on the Web page. The following sections describe how to create custom host components and widgets. For further information, see Appendix A, "HATS Toolkit files," on page 117.

Notes:

1. HATS automatically updates your class loader policy when you add custom components or widgets to your HATS application. This updates the default WAR class loader policy of your HATS application to **Single class loader for application**.
2. If you are using a bidirectional code page, you can control the direction of widgets and other presentation aspects. See Chapter 11, "Using the HATS bidirectional API," on page 111.

HATS component tag and attributes

HATS creates a JavaServer Pages (JSP) page (and writes information to a .jsp file) that reflects host component and widget selections that were made during transformation configuration in the HATS Toolkit. HATS writes this configuration as a set of attributes for the <HATS:Component> tag. The <HATS:Component> tag invokes the code to define host components and specify widget output. To view or edit the transformations in your HATS project, expand the project in the HATS Projects view and look under **Web Content/Transformations**.

This JSP code example shows the format of the <HATS:Component> tag.

```
<HATS:Component type="<fully qualified component class name>"
   widget="<fully qualified widget class name>"
   row="1" col="1" erow="24" ecol="80" componentSettings="" widgetSettings=""
   textReplacement=""  />
```

The attribute data of the <HATS:Component> tag determine which host component and widget classes to call and how to present the widget in HTML output. The **type** attribute of the <HATS:Component> tag specifies which component recognition class to use when recognizing the selected host screen component. The **widget** attribute specifies which rendering class to use when generating the HTML output. The following list describes the other attributes:

row The starting row position for host component recognition.

col The starting column position for host component recognition.

erow The ending row position for host component recognition. You can specify -1 to mean the last row on the host screen.

ecol The ending column position for host component recognition. You can specify -1 to mean the last column on the host screen.

componentSettings

A set of key and value pairs that is sent to the component class. When you specify componentSettings values, specify them in the form `key:value`. If you specify more than one `key:value` pair, separate them with a split vertical bar (|). For example, `<.. componentSettings="key1:value1|key2:value2" ... >`.

You can use the componentSettings attribute for advanced customization of the component. Surround the entire list with quotation marks. For example, if your command line uses the token >>> instead of ==>, you can pass this component setting value here.

widgetSettings

A set of key and value pairs that is sent to the widget class. When you specify widgetSettings values, specify them in the form `key:value`. If you specify more than one `key:value` pairs, separate them with a split vertical bar (|). Surround the entire list with quotation marks. For example, `<... widgetSettings="key1:value1|key2:value2" ... >`.

You can use the widgetSettings attribute for advanced customization of the widget. For example, if you want a table to have a certain number of columns, you can pass this widget setting here.

textReplacement

Any settings that are defined for text replacement for this use of the component. Text replacement attributes are the same as those for definition of text replacement at the project level. See "<replace> tag" on page 127 for descriptions of the attributes.

When defining text replacement, if you want to replace certain special characters, you must use the following in the settings:

@vb. | (split vertical bar)

@cm. , (comma)

@dq. " (double quote)

@lt. < (less than)

@gt. > (greater than)

@eq. = (equal sign)

For example, to replace the string *ABC* with the string *"ABC"*, specify text replacement as follows:

```
textreplacement="ABC=@dq.ABC@dq."
```

HATS performs the following steps to process each <HATS:Component> tag that it encounters in the JSP page.

1. HATS attempts to instantiate the component specified by the **type** attribute of the tag. This attribute can be the fully qualified class name of a component provided by HATS or one created by you.

   Note: You must specify the fully qualified path, e.g. com.company.division.product.*MyComponent*. If you want to be able to

work with your custom component in HATS Toolkit, place the class file in either the WEB-INF/classes directory or in a jar file in the WEB-INF/lib directory. If you prefer, you can import the source (.java) file into the project's source directory. If you have tested your custom component and you do not wish to work with it in HATS Toolkit, you must make the .jar file available to your HATS application when it is installed on WebSphere® Application Server.

2. HATS attempts to instantiate a widget object specified by the **widget** attribute of the tag.

   Note: You must specify the fully qualified path, like `com.company.division.product.MyWidget`. If you want to be able to work with your custom widget in HATS Toolkit place the class file in either the WEB-INF/classes directory or in a jar file in the WEB-INF/lib directory. If you prefer, you can import the source (.java) file into the project's source directory. If you have tested your custom widget and you do not wish to work with it in HATS Toolkit, you must make the jar file available to your HATS application when it is installed on WebSphere Application Server.

3. HATS invokes the `recognize()` method of the component object.

   Each component has a different implementation of the `recognize()` method. The `recognize()` method performs pattern recognition logic and host screen data location. Component classes return an array of com.ibm.hats.transform.ComponentElement objects. This array is the logical representation of what was recognized by this component at the specified region with the specified settings.

4. HATS performs text replacement by calling the `doTextReplacement()` method on each ComponentElement object in the array that was returned by the `recognize()` method of the component object.

5. HATS invokes the `drawHTML()` method of the widget. If an applicable `drawHTML()` method cannot be found, the widget's `draw()` method is called. This method simply returns (unless overridden by subclasses) the concatenation of the toString() calls on each component element in the component element array returned by the component.

   The `drawHTML()` method renders the array of component elements as HTML output. The `drawHTML()` method does this by returning a StringBuffer object that contains its rendering of the supplied component element array. The widget used for the component determines how the component elements are rendered. Two widgets can take the same type of input. For example, the Link widget and the Button widget can both take the same type of input, but their `drawHTML()` methods generate different HTML content.

The following figure illustrates the flow of data from a region of the host screen, through the component and widget, to the Web page.

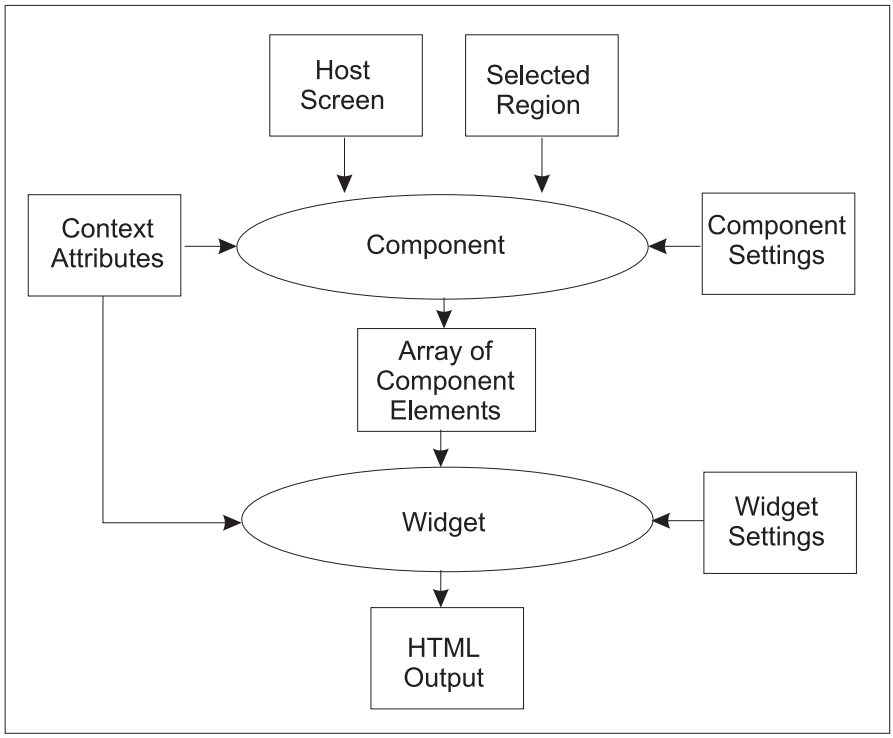


Figure 1. Flow of data from host screen to Web page

Creating a custom host component

HATS provides a Create Component wizard to help you create custom components. You can start the wizard in several ways:

- From the **File > New > HATS Component** menu in HATS Toolkit
- From the **HATS > New > Component** menu in HATS Toolkit
- From the context (right-click) menu of a HATS project, select **New HATS > Component**

There are two panels of the Create Component wizard. On the first panel, you provide the name of the project, the name of the new component, and the name of the Java package for the component. Optionally, you can select a check box to include stub methods that allow you to define a GUI panel for configuring the settings used by the new component (see "HATS Toolkit support for custom component and widget settings" on page 26 for more information). On the second panel, you enter the name you want displayed for the new component in the HATS Toolkit and select the widgets to associate with the component. Widget association is not necessary to complete the wizard. You can define the association of components and widgets later. Refer to "Registering your component or widget" on page 25 for more information about associating components and widgets.

The following sections explain the required elements of a custom component that the wizard provides:

- Extends the host component abstract class, `com.ibm.hats.transform.components.Component`.

If one of the HATS host components is very similar to what you need, it will be easier to extend that component. See "Extending component classes" on page 22 for more information.

- Adds the constructor method. This method, named for your component, must accept a com.ibm.hats.common.HostScreen object. For example:

```
public MyComponent(HostScreen hostScreen) {
  super(hostScreen);
}
```

  The constructor should initialize parameters that the `recognize()` method will require, based on the host screen object.

- Adds the `recognize()` method.

```
public ComponentElement[] recognize(BlockScreenRegion region,
                                    Properties settings)
```

  The `recognize()` method has a different implementation in each host component class. It accepts the region and settings passed to it and returns an array of component element objects. You should implement this method to implement your own pattern recognition logic.

  The `recognize()` method must return an array of ComponentElement objects, as defined in com.ibm.hats.transform.elements.ComponentElement. Each HATS component returns a slightly different set of elements that extend ComponentElement. For example, the SelectionListComponent returns an array of SelectionComponentElement objects. This array of component elements is passed to the specified widget, so be sure to return an array of elements that can be accepted by the widget you want to use.

  For a description of the arguments of this method, refer to the HATS API References (Javadoc) for the `recognize()` method of the Component class. See "Using the API documentation (Javadoc)" on page 2.

- Adds the source code for the component into the **Source** folder of your project
- Compiles the new component .java file, if you have **Build Automatically** checked in the Rational SDP workbench preferences (**Window > Preferences > General > Workspace** or **Project > Build Automatically**). If the component is not compiled into a .class file, it is not available for use in the HATS Toolkit.
- Registers the new component in the `ComponentWidget.xml` file. See "Registering your component or widget" on page 25 for more information about registering components.

If you selected the check box to include HATS Toolkit graphical user interface support methods, enabling you to modify the settings of the component, the Create Component wizard adds the following methods:

- Method to return the number of pages in the property settings:

```
public int getPropertyPageCount() {
  return (1);
}
```

- Method to return the settings that can be customized:

```
public Vector getCustomProperties(int iPageNumber, Properties properties,
   ResourceBundle bundle) {
  return (null);
}
```

- Method to return the default values of the settings that can be customized:

```
public Properties getDefaultValues(int iPageNumber) {
  return (super.getDefaultValues(iPageNumber));
 }
```

See “HATS Toolkit support for custom component and widget settings” on page 26 for more information about the methods necessary to support your custom component.

Note: If you want your component to work properly within Default Rendering, you must set the consumed region (that is, the area of the host screen that has been processed) on each component element that your component returns, before returning the component element. This tells the Default Rendering that this region of the screen has been consumed, or processed, by a host component and should not be processed again. To set the consumed region, use this method:

```
public void setConsumedRegion(BlockScreenRegion region)
```

Refer to the HATS API References (Javadoc) for the ComponentElement class for more information. See “Using the API documentation (Javadoc)” on page 2.

Extending component classes

HATS provides a number of host component classes. You can extend any of the host component classes that are found in the `ComponentWidget.xml` file by replacing the statement `public class` *`MyCustomComponent`* `extends Component` in the created .java file for the new component with the class name of an existing component. For example:

```
public class MyCustomComponent
             extends com.ibm.hats.transform.components.CommandLineComponent
```

Note: Bidirectional components are stored in the com.ibm.hats.transform.components.BIDI package. The names of bidirectional classes for components are the same as regular components, but they are followed by “BIDI”; for example, com.ibm.hats.transform.components.BIDI.CommandLineComponentBIDI.

Each HATS component performs recognition of elements of the host screen in the `recognize()` method. To extend a host component and accomplish the specific recognition task you need, you can use either of these approaches:

- Extend one of the component classes that is provided by HATS and override the `recognize()` method of the component. Somewhere in your `recognize()` method you should add a call like `super.recognize(region, settings);` to invoke the `recognize()` method of the class you extended. You can modify the process by changing the settings before calling the superclass, or by manipulating the output returned by the superclass.
- Extend one of the component classes that is provided by HATS and override the `recognize()` method of the component. Instead of using the `recognize()` method of the superclass, invoke the `recognize()` method of one of the other component classes. This approach will be useful if you want to recognize a complex host component that combines aspects of more than one of the HATS components.

The Create Component wizard generates a recognize() method that returns null, which indicates that the host screen region is not recognized by the new component. To change the custom component to act as the HATS component it is extended from, whose elements contain all of the correct ComponentElements, remove the "return null" from the .java file and change the code in the component code. For example:

```
public ComponentElement[] recognize(LinearScreenRegion region, Properties settings) {
ComponentElement [] elements = super.recognize(region, settings);
return elements;
}
```

HATS instantiates the custom component based on the setting of the **type** attribute of the <HATS:Component> tag.

To edit the `ComponentWidget.xml` file, click the **Navigator** tab of the HATS Toolkit. The `ComponentWidget.xml` file is shown at the bottom of the **Navigator** view of your project. See "Registering your component or widget" on page 25 for more information about the `ComponentWidget.xml` file.

Creating a custom HTML widget

HATS provides a Create Widget wizard to help you create custom widgets. You can start the wizard in several ways:

- From the **File > New > HATS Widget** menu in HATS Toolkit
- From the **HATS > New > Widget** menu in HATS Toolkit
- From the context (right click) menu of a HATS project, select **New HATS > Widget**

There are two panels in the Create Widget wizard. On the first panel, you provide the name of the project, the name of the new widget, and the name of the Java package for the widget. Optionally, you can select a check box to include stub methods that allow you to define a GUI panel for configuring the settings used by the new widget (see "HATS Toolkit support for custom component and widget settings" on page 26 for more information). On the second panel, enter the name you want displayed for the new widget in the HATS Toolkit and select the components to associate with the widget.

The wizard provides the following required elements of a custom widget:

- Extends the widget abstract class and implements the HTMLRenderer interface: `public class MyCustomWidget extends Widget implements HTMLRenderer`.

  See "Extending widget classes" on page 24 for more information.
- Adds the constructor method:

  ```
  public MyCustomWidget(ComponentElement[] arg0, Properties arg1) {
    super(arg0, arg1);
  }
  ```
- Adds the following method to generate HTML for Web project output.

  ```
  public StringBuffer drawHTML() {
     StringBuffer buffer = new StringBuffer(256);
     HTMLElementFactory factory = HTMLElementFactory.newInstance(
       contextAttributes, settings);
     return (buffer);
    }
  ```

  The HTMLElementFactory class automatically generates any JavaScript needed to present the widget. Refer to the HATS API References (Javadoc) for detailed information about the HTMLElementFactory class and more examples of its use. See "Using the API documentation (Javadoc)" on page 2.
- Adds the source code for the widget into the **Source** folder of your project
- Compiles the new widget .java file, if you have **Build Automatically** selected in the Rational SDP workbench preferences (**Window > Preferences > General > Workspace**) or the Project menu. If the widget is not compiled into a .class file, it is not available for use in the HATS Toolkit.

- Registers the new widget in the `ComponentWidget.xml` file. See "Registering your component or widget" on page 25 for more information about registering widgets.

If you selected the check box to include HATS Toolkit graphical user interface support methods, enabling you to modify the settings of the widget, the Create Widget wizard adds the following methods:

- Method to return the number of pages in the property settings:

```
public int getPropertyPageCount() {
  return (1);
}
```

- Method to return the settings that can be customized:

```
public Vector getCustomProperties(int iPageNumber, Properties properties,
   ResourceBundle bundle) {
  return (null);
}
```

- Method to return the default values of the settings that can be customized:

```
public Properties getDefaultValues(int iPageNumber) {
  return (super.getDefaultValues(iPageNumber));
 }
```

See "HATS Toolkit support for custom component and widget settings" on page 26 for more information about the methods necessary to support your custom widget.

Extending widget classes

HATS provides a number of widget classes. You can extend any of the widget classes found in the `ComponentWidget.xml` file by replacing the

```
public class MyCustomWidget extends Widget implements HTMLRenderer
```

in the created .java file for the new widget with the class name of an existing widget, such as

```
public class MyCustomWidget extends
  com.ibm.hats.transform.widgets.FieldWidget
```

Note: Bidirectional widgets are stored in the com.ibm.hats.transform.widgets.BIDI package. The names of bidirectional classes for widgets are the same as regular widgets, but they are followed by "BIDI"; for example,

```
com.ibm.hats.transform.widgets.BIDI.FieldWidgetBIDI
```

If you want to modify an existing widget, you must extend one of the existing widget classes and override its `drawHTML` method. Refer to the HATS API References (Javadoc) for details about widget interfaces and methods. See "Using the API documentation (Javadoc)" on page 2.

HATS instantiates the custom widget based on the setting of the **widget** attribute of the <HATS:Component> tag.

Widgets and global rules

Widgets that present input fields should check whether the input field has already been processed by a HATS global rule. When a host screen is received, HATS searches it for host components that match global rules that are defined for that HATS application. When your widget checks whether the input field has already been processed by a HATS global rule, the call returns null if the input field has not been processed. If the input field has already been processed according to a global rule, the call returns the transformation fragment to which the input field

has been transformed by the global rule. Your widget should output the fragment rather than processing the component element. Examples are shown below for Web applications:

```
String ruleReplacement =
  RenderingRulesEngine.processMatchingElement(componentElement, contextAttributes);
if (ruleReplacement != null) {
  buffer.append(ruleReplacement);
} else {
    .
    .
    .
}
```

Add the above example to the drawHTML() method for the widget.

Registering your component or widget

Registering your custom components and widgets in the `ComponentWidget.xml` file makes them available for use in the HATS Toolkit, such as in the Insert Host Component wizard.

Host components must map to specific widgets. Custom host components can map to any existing widget or to a custom widget. The Create a custom component or widget wizards register your custom components and widgets in the `ComponentWidget.xml` file, and associates components and widgets. When using the wizards, if you did not associate your custom component or widget, you need to edit the `ComponentWidget.xml` file and add the associations. To edit the `ComponentWidget.xml` file, click the **Navigator** tab of the HATS Toolkit. The `ComponentWidget.xml` file is shown at the bottom of the **Navigator** view of your project.

Note: If you decide to delete a custom component or widget after it has been registered, simply deleting the source code for the component or widget from the **Source** folder of your project is not enough to completely remove it. It is still referenced in the registry and there is no programmatic way to remove it. You should remove it from the registry by editing the `ComponentWidget.xml` file and deleting the references to the component or widget.

Following is an example of the `ComponentWidget.xml` file that shows the HATS-supplied Field Table component and one of the associated widgets, the vertical bar graph widget.

```
<ComponentWidgetList>
  <components>
    <component className="com.ibm.hats.transform.components.FieldTableComponent"
       displayName="Field table" image="table.gif">
      <associatedWidgets>
        <widget className="com.ibm.hats.transform.widgets.VerticalBarGraphWidget"/>
      </associatedWidgets>
    </component>
  </components>


  <widgets>
    <widget className="com.ibm.hats.transform.widgets.VerticalBarGraphWidget"
       displayName="Vertical graph" image="verticalBarGraph.gif" />
  </widgets>
</ComponentWidgetList>
```

As you can see, there are two sections to this file: components and widgets.

The components section contains the list of all registered components. To register a custom component and make it available to the HATS Toolkit, add a <component> tag and the associated <widget> tags to the `ComponentWidget.xml` file. You must supply a className, displayName, and the associated widgets.

className
Identifies the Java class that contains the code to recognize elements of the host screen. The class name is usually in the form com.*myCompany.myOrg.ClassName*.

displayName
Identifies the name by which your custom component is known and how it appears in the list of components in the HATS Toolkit. This name must be unique among the registered components. The form of the displayName for a custom component is simply a string. Spaces are allowed in the displayName.

image The image attribute identifies the image to use for your component when it appears in the HATS Toolkit.

widget
Identifies the widgets that are associated with this component. There must be a separate <widget> tag for each associated widget. All of the <widget> tags for the component must be defined within the <associatedWidgets> tag and its </associatedWidgets> ending tag. The <widget> tag within the <associatedWidgets> tag contains only the className attribute, which identifies the Java class that contains the code to link the widget to the component. The class name is usually in the form com.*myCompany.myOrg.ClassName*.

The widgets section contains the list of all registered widgets. To register a widget, link it to a component, make it available for use in the HATS Toolkit, and add a <widget> tag to the `ComponentWidget.xml` file. You must supply a className and a displayName.

className
Identifies the Java class that contains the code to render the widget. The class name is usually in the form com.*myCompany.myOrg.ClassName*.

displayName
Identifies the name by which your custom widget is known and how it appears in the list of widgets in the HATS Toolkit. This name must be unique among the registered widgets. The form of the displayName for a custom widget is simply a string. Spaces are not allowed in the displayName. However, you can use an underscore (_) in place of a space.

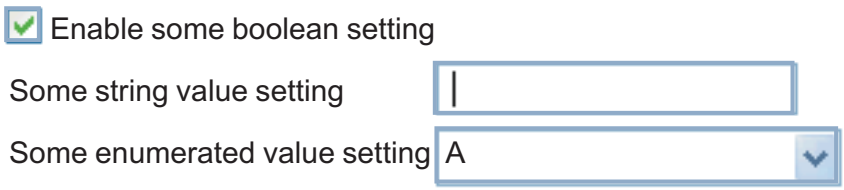
HATS Toolkit support for custom component and widget settings

You can provide GUI support for modifying the settings of your custom component and widget. This is useful if other developers will be using your custom component or widget or you want to easily test different combinations of settings using the preview features available in the HATS Toolkit. The base component and widget classes implement the ICustomPropertySupplier interface. This interface allows a component or widget to contribute setting information to the HATS Toolkit. This information is used to render a panel by which the settings of the component or widget can be modified. Not all settings need to be exposed in the GUI.

The `getCustomProperties()` method returns a vector of **HCustomProperty** customizable property objects. Each **HCustomProperty** object represents a setting of the component or widget. The HATS Toolkit renders each **HCustomProperty** objects based on its type. For example, an object of type HCustomProperty.TYPE_BOOLEAN is rendered as a GUI checkbox.

The following sample code demonstrates how a widget can provide GUI support for three of its settings (mySetting1, mySetting2, and mySetting3):

```
// Returns the number of settings panels (property pages) to be contributed
//by this widget.  The returned value must be greater than or equal to 1 if
//custom properties will be supplied via the getCustomProperties() method.
public int getPropertyPageCount() {
         return 1;
}

// Returns a Vector (list) of custom properties to be displayed in the GUI
//panel for this component or widget.
public Vector getCustomProperties(int iPageNumber, Properties properties,
         ResourceBundle bundle) {
         Vector props = new Vector();

// Constructs a boolean property that will be rendered as a checkbox
HCustomProperty prop1 = HCustomProperty.new_Boolean("mySetting1",
                "Enable some boolean setting", false, null, null);
props.add(prop1);

// Constructs a string property that will be rendered as a text field
HCustomProperty prop2 = HCustomProperty.new_String("mySetting2",
                "Some string value setting", false, null,
                 null, null, null);
props.add(prop2);

// Constructs an enumeration property that will be rendered as a drop-down
HCustomProperty prop3 = HCustomProperty.new_Enumeration("mySetting3",
                 "Some enumerated value setting", false,
                 new String[] { "A", "B", "C" }, new String[]
                 { "Option A", "Option B", "Option C" }, null, null, null);
props.add(prop3);

return props;     }
```



The values supplied by the user of the custom component or widget will be available in the *componentSettings* **Properties** object passed into the `recognize()` method of the component or the *widgetSettings***Properties** object passed into the constructor of the widget. The `getCustomProperties()` method may be called during runtime to collect default values for settings.

For a description of the arguments and usage of these methods, refer to the HATS API References (Javadoc) for the HCustomProperty class. See “Using the API documentation (Javadoc)” on page 2.

Chapter 4. Working with Dojo widgets

Customizing a HATS Dojo widget

When you first add a host component rendered by a HATS Dojo widget to a transformation .jsp file, the component and widget selections are defined within a <HATS:Component> tag. For more information about the <HATS:Component> tag, see "HATS component tag and attributes" on page 17. If the HATS default widget definitions meet you needs, then this is all that is required.

The example below shows a Command line component rendered by a HATS default Dojo Combo box widget.

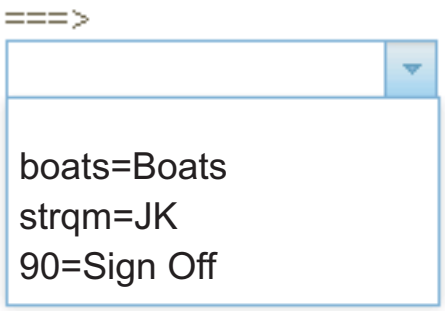


Figure 2. HATS default Dojo Combo Box widget

In the transformation .jsp file this default component and widget are defined within a <HATS:Component> tag as shown below.

```
<HATS:Component
   type="com.ibm.hats.transform.components.CommandLineComponent"
   componentSettings=""
 row="20" col="7" erow="20" ecol="21"
   alternate=""
 widget="com.ibm.hats.transform.widgets.dojo.ComboBoxWidget"
   widgetSettings="stringListItems:Boats=boats;JK=strqm;Sign Off=90;
    |autoSubmitOnSelect:false|useString:true|"
 alternateRenderingSet=""  textReplacement="" />
```

If you want to customize a HATS Dojo widget to add function in addition to what is provided using the defaults, you must first transform the component and widget definitions. To do this, select the <HATS:Component> tag, right-click, and select **HATS Tools > Transform for Dojo Editing**. The Transform for Dojo Editing wizard transforms the <HATS:Component> tag to a <HATS:Render> tag. The example below shows the <HATS:Component> tag above after being transformed to a <HATS:Render> tag.

```
<HATS:Render

<!-- Start of component settings -->
type="com.ibm.hats.transform.components.CommandLineComponent"
componentSettings=""
row="20" col="7" erow="20" ecol="21"
textReplacement="">
```

```
<!-- End of component settings -->

<!-- Start of ComboBoxWidget -->
<!-- com.ibm.hats.transform.widgets.dojo.ComboBoxWidget -->

<!-- Start of JSON data source -->

<!-- Start of JSON data for the component element -->
<script>var HATSJSON_<HATS:ElementId/> = <HATS:JSON/>;</script>
<!-- End of JSON data for the component element -->

<!-- Start of JSON widget settings -->
<script>var DOJOWidgetSettings_<HATS:ElementId/> =
{"type":"ComboBoxWidget",
"value":{"stringListItems":"Boats=boats;JK=strqm;Sign Off=90;",
         "autoSubmitOnSelect":"false","useString":"true"
}
};
<!-- End of JSON widget settings -->
</script>
<!-- End of JSON data source -->

<!-- Start of rendered widget -->
<div id="<HATS:ElementId/>">
<label for id="<HATS:ElementId/>_input"
           id="<HATS:ElementId/>_label"></label>
<input id="<HATS:ElementId/>_input"></input>
</div>
<!-- End of rendered widget -->

<!-- Start of data binding -->
<script type="text/javascript"
        src="../common/hatsdojo/hsr_comboboxwidget.js">
</script>
<script type="text/javascript">
if (HATSJSON_<HATS:ElementId/> && (HATSJSON_<HATS:ElementId/>.value){
   dojo.addOnLoad(function(){
   var uLabel = dojo.byId("<HATS:ElementId/>_label");
   var jsonData = (HATSJSON_<HATS:ElementId/>;
   var widgetSettings = DOJOWidgetSettings_<HATS:ElementId/>;
  var jsonList = getListItemsFromJSONData(jsonData,
     getListItemsFromHATSWidgetSettings(widgetSettings));
 var storeList = new dojo.data.ItemFileReadStore(
          {data: {identifier:"value",
                  items:createUniqueItemsList(jsonList,"value")
              }
          }
);
 var uComboBoxWidget = new dijit.form.ComboBox(
         {name:getPosLengthStringFromJSONData(jsonData),
          store:storeList,searchAttr:"fullName"
         },
          "<HATS:ElementId/>_input"
);
  //load the JSON information and behavior into the Widget
  bindJSONDataToComboBox(uLabel, uComboBoxWidget,
        jsonData, widgetSettings);
  setInputFieldFocus();
   });
 }
```

```
</script>
<!-- End of data binding -->

<!-- /com.ibm.hats.transform.widgets.dojo.ComboBoxWidget -->
<!-- End of ComboBoxWidget -->

</HATS:Render>
```

Notice in the example above, comments have been added to separate different sections of the <HATS:Render> tag. These sections are described below.

Component settings

These settings are the same as on the <HATS:Component> tag. For descriptions, see "HATS component tag and attributes" on page 17.

Widget settings

The widget settings are contained in a JavaScript Object Notation (JSON) object and used in the creation and binding of the Dojo widget in JavaScript for behaviors, options, and default values.

`<HATS:ElementId/>` creates a unique identity for the rendered component element at runtime.

`<HATS:JSON/>` creates the actual JSON object at runtime.

JSON data source

The JSON data source is created from the ComponentElements using a toJSON() method. This JSON data is used to bind the Dojo widget to HATS data. More information can be found in the APIs for the ComponentElements. See "Using the API documentation (Javadoc)" on page 2.

Rendered widget

The rendered widget is the base HTML and JavaScript that is changed at Dojo load time to create a HATS Dojo widget.

Data binding

The data binding uses the settings and data provided to change the rendered widget into a HATS Dojo widget. This typically includes providing the Dojo widget event behaviors such as focus and selection, default values, and any options, settings, or data necessary for correct operation and layout.

HATS Dojo widget customization examples

The following sections include examples of customized HATS Dojo widgets. For more information about these and other Dojo widgets, see the Dojo Toolkit API documentation at http://dojotoolkit.org/api/.

Combo box

This example shows various ways to customize a HATS Dojo Combo box widget.

Start by creating a Combo box widget, for example, for the `From City` input field on the following host screen for the Flights Reservation System application.

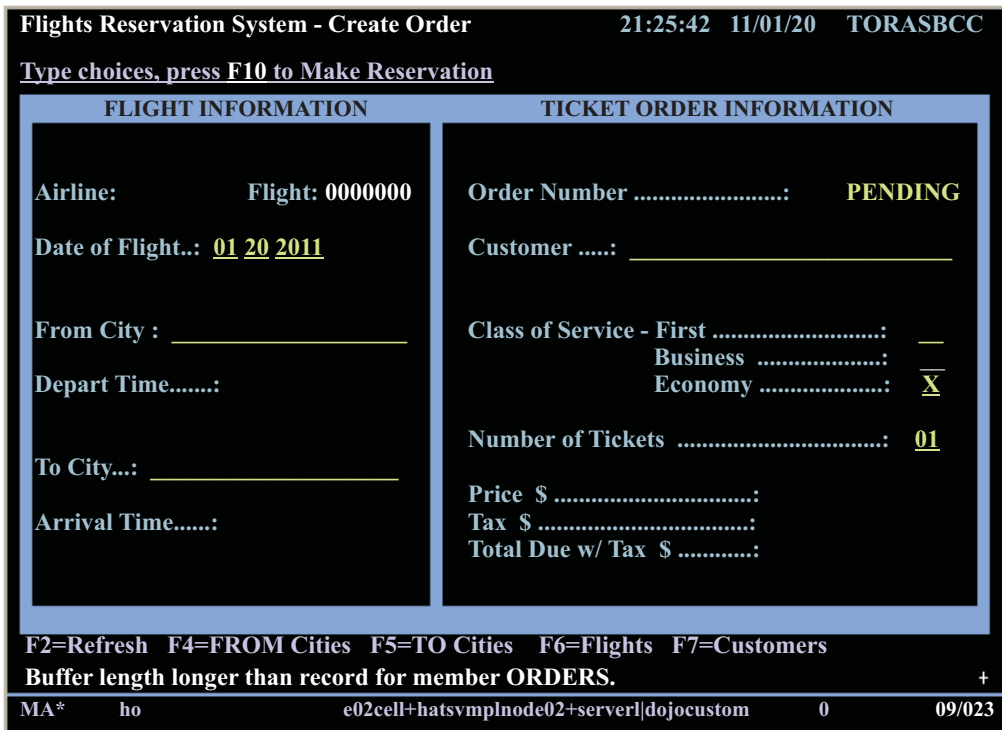


Figure 3. Host screen for the Flights Reservation System

When creating the Combo box widget use the **Fill from string** setting to specify the **List items** to fill the drop-down list. In this example, use the following string of city names:

```
Albany;Albuquerque;Atlanta;Boston;Chicago;Dallas;Los Angeles;Miami;
Montreal;Raleigh;Rochester, MN;Salt Lake City;San Diego;Toronto;Vancouver;Washington DC
```

After you create the Input field component rendered with the Combo box widget, the widget appears on the transformation .jsp file as shown below:

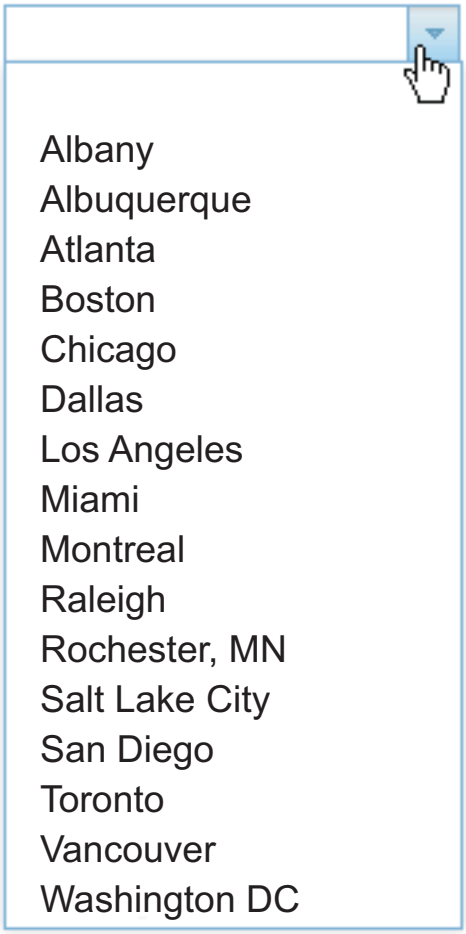


Figure 4. Dojo Combo box widget with list of cities

When you preview or run this example, type the letters, `al`, into the Combo box. Notice that all of the cities that contain the letters, `al`, are displayed in the drop-down list.

Figure 5. Dojo Combo box widget with filtered list of cities

To customize the Combo box widget so that only cities that start with the typed letters are display in the drop-down list, edit the transformation .jsp file and perform the following steps:

1. Locate the <HATS:Component> tag in the transformation .jsp file. Below is the source code that is created for this example.

```
<HATS:Component
    type="com.ibm.hats.transform.components.InputComponent"
      componentSettings="" textReplacement=""
      row="12" erow="12" col="17" ecol="32"
   widget="com.ibm.hats.transform.widgets.dojo.ComboBoxWidget"
     widgetSettings="stringListItems:Albany;Albuquerque;Atlanta;Boston;Chicago;Dallas;
      Los Angeles;Miami;Montreal;Raleigh;Rochester, MN;Salt Lake City;San Diego;
      Toronto;Vancouver;Washington DC|
      autoSubmitOnSelect:false|useString:true|" alternate="" alternateRenderingSet="" />
```

2. Select the <HATS:Component> tag, right-click, and select **HATS Tools > Transform for Dojo Editing**. This transforms the <HATS:Component> tag to a <HATS:Render> tag.
3. Referring to the Dojo API, for example, at http://dojotoolkit.org/api/1.5/dijit/form/ComboBox, you see that the `queryExpr` property controls how to match entries in the drop-drop list. The expression, `${0}*`, can be used to display only list entries that start with the characters that the user types. Within the <HATS:Render> tag, notice that the widget variable is named, uComboBoxWidget. To customize your widget to display only cities that start with the typed letters, following the setInputFieldFocus(); statement add the statement shown below:

```
setInputFieldFocus();
uComboBoxWidget.queryExpr = "\${0}*";
```

   Note: Notice the backslash (\) before the queryExpr property `${0}*`. This prevents the JSP translator from processing it as Expression Language (EL) syntax.

Now when you preview or run this example, and type the letters, `al`, into the Combo box, notice that only cities that start with these letters are displayed in the drop-down list.

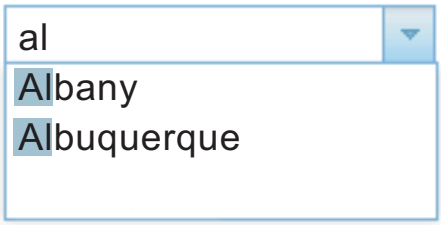


Figure 6. Dojo Combo box using starting-with list of cities

The Combo box widget also supports validation of user-supplied input. Again, referring to the Dojo API documentation, you see that the `regExp` property can be used to restrict the format of user-supplied input. For example, if you want to specify that no numbers are allowed in the user's input, following the statement you added in the previous example, add the statement shown below:

```
uComboBoxWidget.regExp = "[^0-9]*";
```

In addition, if you want to add a prompt message and change the text of the default invalid value message , add the final two statements shown below:

```
setInputFieldFocus();
uComboBoxWidget.queryExpr = "\${0}*";
uComboBoxWidget.regExp = "[^0-9]*";
uComboBoxWidget.promptMessage = "Please enter the departure city.";
uComboBoxWidget.invalidMessage = "Numeric characters are not allowed.";
```

Now when you preview or run this example, notice your prompt message is displayed when you position the cursor in the combo box.

Figure 7. Dojo Combo box prompt message

Type a value containing a number. Notice your invalid message is displayed, and the combo box changes color and displays a warning icon.

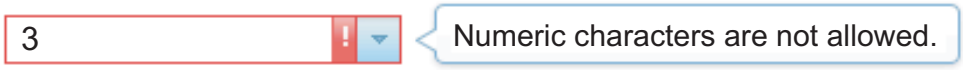


Figure 8. Dojo Combo box invalid message

Enhanced grid

This example shows how to change the HATS Enhanced grid widget to add a single-selection radio button for each row in the table.

Start by creating an Enhanced grid widget, for example, for a table similar to the one shown on the following host screen.

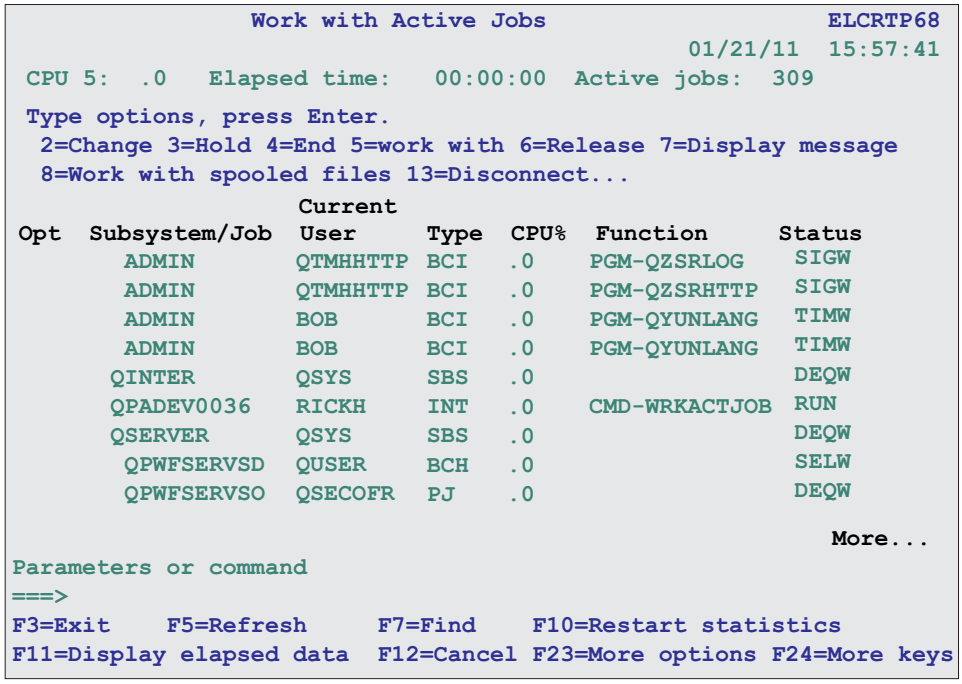


Figure 9. Work with Active Jobs host screen

After you create a Table component rendered with an Enhanced grid widget, the widget appears on the transformation .jsp file as shown below.

Note: This figure shows the widget appearance when using the Dojo Claro theme. The widget might appear differently when using a different Dojo theme.

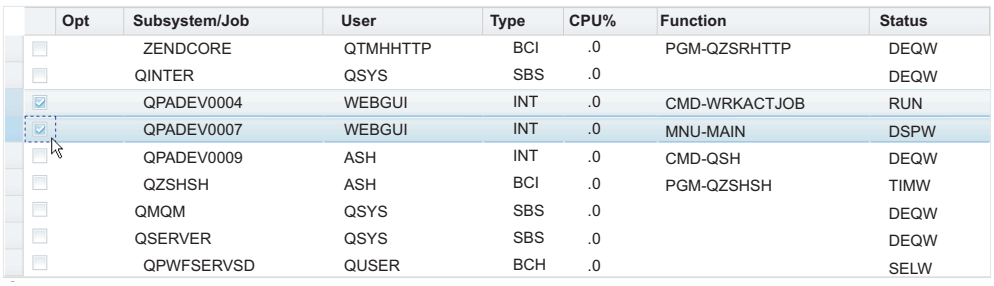


Figure 10. Dojo Enhanced grid widget

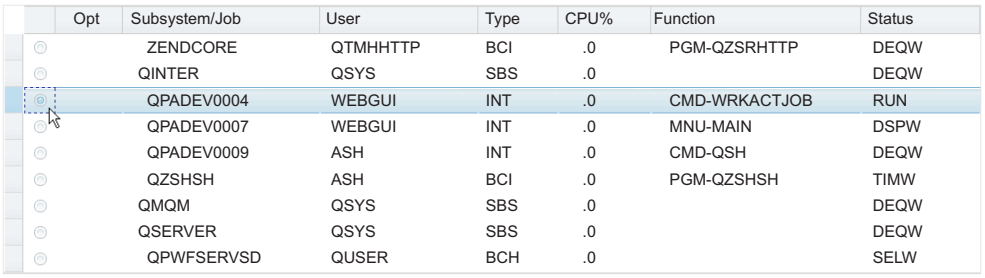
To change the indirectSelection plug-in to provide a single-selection radio button for each row, edit the transformation .jsp file and perform the following steps:

1. Locate the <HATS:Component> tag in the transformation .jsp file. Select the tag, then right-click and select **HATS Tools > Transform for Dojo Editing**. This transforms the <HATS:Component> tag to a <HATS:Render> tag.
2. Locate and change the `grid` variable as shown below:

```
var grid = new dojox.grid.EnhancedGrid(
  {
   autoWidth: true,
   autoHeight: true,
   structure: tableHeader,
   plugins: {nestedSorting: true, dnd: true, indirectSelection: true},
```

```
        selectionMode: "single"
    },
    document.createElement('div')
);
```

The Enhanced grid widget now appears on the transformation .jsp file as shown below. In this example, single-selection radio buttons are added to each row in the table. The selected row can be dragged to another location in the table.



Opt	Subsystem/Job	User	Type	CPU%	Function	Status
	ZENDCORE	QTMHHTTP	BCI	.0	PGM-QZSRHTTP	DEQW
	QINTER	QSYS	SBS	.0		DEQW
	QPADEV0004	WEBGUI	INT	.0	CMD-WRKACTJOB	RUN
	QPADEV0007	WEBGUI	INT	.0	MNU-MAIN	DSPW
	QPADEV0009	ASH	INT	.0	CMD-QSH	DEQW
	QZSHSH	ASH	BCI	.0	PGM-QZSHSH	TIMW
	QMQM	QSYS	SBS	.0		DEQW
	QSERVER	QSYS	SBS	.0		DEQW
	QPWFSERVSD	QUSER	BCH	.0		SELW

Figure 11. Dojo Enhanced grid widget with single row select

Filtering select

This example shows how to fill the drop-down list of a HATS Dojo Filtering select widget from a global variable.

Start by creating a Filtering select widget, for example, for the `Account` number input field on the following host screen for the Accounts application.

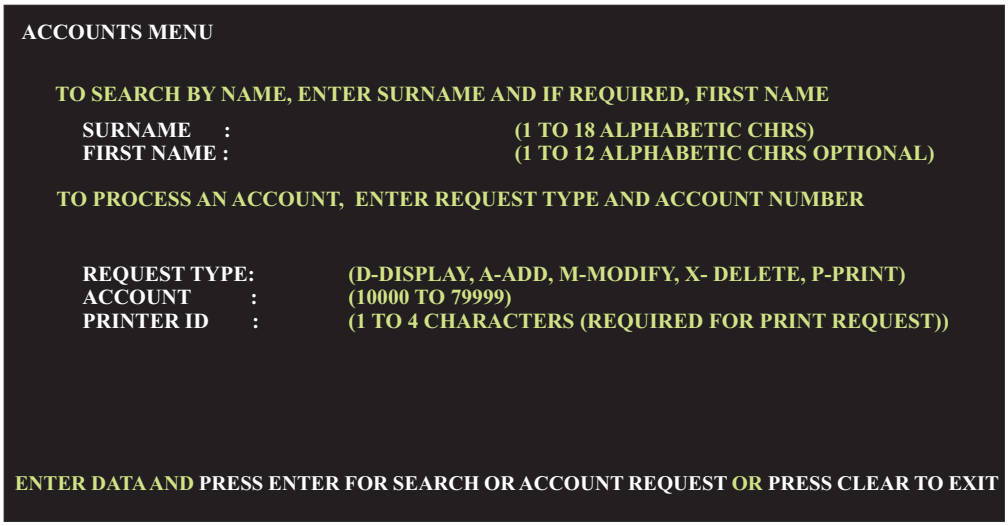


Figure 12. Host screen with account number input field

After you create an Input field component rendered with a Filtering select widget, the widget appears on the transformation .jsp file as shown below:

Figure 13. Dojo Filtering select widget

Edit the transformation .jsp file and perform the following steps:

1. Locate the <HATS:Component> tag in the transformation .jsp file. Below is the source code that is created for this example.

```
<HATS:Component
   type="com.ibm.hats.transform.components.InputComponent"
   componentSettings="" textReplacement=""
      row="11" erow="11" col="22" ecol="26"
   widget="com.ibm.hats.transform.widgets.dojo.FilteringSelectWidget"
   widgetSettings="" alternate="" alternateRenderingSet="" />
```

2. Select the <HATS:Component> tag, right-click, and select **HATS Tools > Transform for Dojo Editing**. This transforms the <HATS:Component> tag to a <HATS:Render> tag.
3. Position the cursor after the last </script> tag in the .jsp source. Right-click and select **HATS Tools > Insert Global Variable**.
4. On the Insert Global Variable window, select the global variable whose contents contains the data to fill the Filtering select drop-down list. In this example, a global variable named `acctnumGV` contains the text, `10011;10012;10013;10014`, which are valid account numbers for this application and are in the same format, separated by a semicolon (;), used by the **List items** setting of the Filtering select widget.
5. One result of using the Insert Global Variable tool is to add the following import statement before the <html> tag in the .jsp source.

```
<%@page import="com.ibm.hats.common.*"%>
<html>
```

6. A second result is to add the following statement at the cursor location, in this case, after the last </script> tag in the .jsp source.

```
<%= ((TransformInfo)request.getAttribute(CommonConstants.REQ_TRANSFORMINFO))
.getGlobalVariable("acctnumGV", true).getString(0) %>
```

7. Within the <HATS:Render> tag, locate the creation of the jsonList variable. Following this statement, add statements to create a gvString variable and then add the items in the gvString variable to the jsonList variable. To initialize the gvString variable, cut and paste the statement added by the Insert Global Variable tool following the </script> tag. When complete, the code should be as shown below.

```
var jsonList = getListItemsFromJSONData(jsonData, getListItemsFromHATSWidgetSettings(widgetSettings));
var gvString = '<%= ((TransformInfo)request.getAttribute(CommonConstants.REQ_TRANSFORMINFO))
 .getGlobalVariable("acctnumGV", true).getString(0) %>';
jsonList = getListItemsFromString(gvString, jsonList);
var storeList = new dojo.data.ItemFileReadStore({data: {identifier:"value",
 items:createUniqueItemsList(jsonList,"value")}});
```

The Filtering select widget now appears on the transformation .jsp file as shown below. In this example, the drop-down list contains the set of valid account numbers provided by the accountNum global variable.

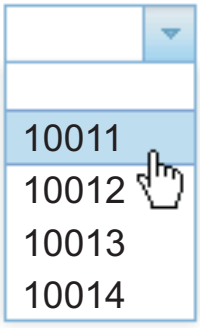


Figure 14. Dojo Filtering select widget using global variable

Number spinner

This example shows how to create a Number spinner Dojo widget to use for the same `Account` number input field used in the Filtering select example. See Figure 12 on page 37.

Start by creating a HATS Text box Dojo widget to render the input field.

After you create an Input field component rendered with a Text box widget, the widget appears on the transformation .jsp file as shown below:

Figure 15. Dojo Text box widget

Edit the transformation .jsp file and perform the following steps:

1. Add a dojo.require statement for the Number spinner widget as shown in the example below.

```
<script type="text/javascript">
dojo.require("dojo.parser");
dojo.require("dijit.form.TextBox");
dojo.require("dijit.form.NumberSpinner");
</script>
```

2. Locate the <HATS:Component> tag in the transformation .jsp file. Below is the source code that is created for this example.

```
<HATS:Component
   type="com.ibm.hats.transform.components.InputComponent"
      componentSettings="" textReplacement="" BIDIOpposite="false"
      row="11" erow="11" col="22" ecol="26"
    widget="com.ibm.hats.transform.widgets.dojo.TextBoxWidget"
      widgetSettings="" alternate="" alternateRenderingSet="" />
```

3. Select the <HATS:Component> tag, right-click, and select **HATS Tools > Transform for Dojo Editing**. This transforms the <HATS:Component> tag to a <HATS:Render> tag.
4. Within the <HATS:Render> tag, locate where the Text box widget is created. Comment out (or remove) the creation of the Text box widget and create a Number spinner widget as shown below.

```
// comment out the original text box code
// var uInputWidget = new
// dijit.form.TextBox({"type":inputType},"<HATS:ElementId/>_input");
```

```
var uInputWidget = new
dijit.form.NumberSpinner({"smallDelta":1,"constraints":
{"min":10011,"max":10037,"places":0},"required":"true"},
"<HATS:ElementId/>_input");
```

The Number spinner widget now appears on the transformation .jsp file as shown below. In this example, the `min` and `max` options are set to match the correct range of account numbers for this particular application.

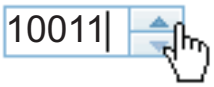


Figure 16. Dojo Number spinner widget

Using the Dojo TabContainer widget

HATS tabbed folder support is deprecated in HATS V9.5. While support for tabbed folders continues for now, IBM reserves the right to remove this capability in a subsequent release of the product. One alternative is to use the TabContainer Dojo Layout widget to create tabs and use HATS widgets to render host components within the tabs.

Using the Dojo TabContainer widget in a HATS Web project

The example that follows shows using the Dojo TabContainer widget in a HATS Web project to render the data on the Display Report host screen in two tabs, one using the HATS Dojo Enhanced grid widget and the other using the HATS Graph widget.

To use the Dojo TabContainer widget in this example, follow these steps:

1. Create a HATS Web project with the **Use Dojo technology** option selected.
2. Start the Host Terminal, navigate to the first screen of the report and click **Create HATS Screen Customization** on the toolbar.

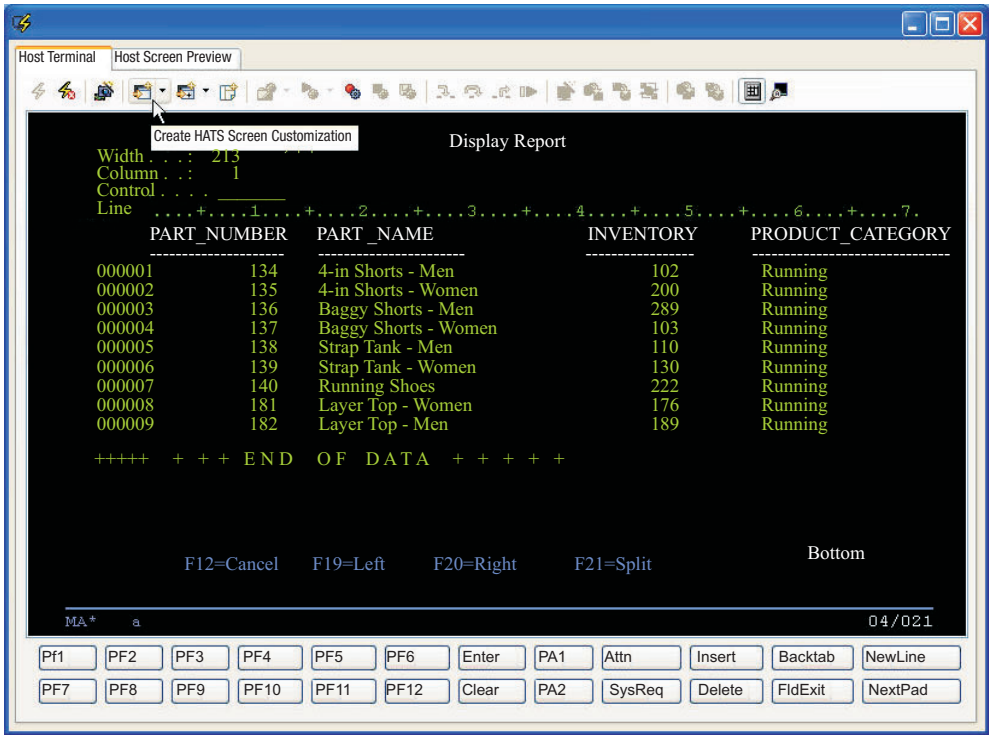


Figure 17. Host screen to render using the TabContainer Dojo widget

3. On the Screen Customization page, accept the defaults and click **Next**.
4. On the Screen Recognition Criteria page, specify how to identify the screen and click **Next**.
5. On the Actions page, accept the defaults and click **Finish**. The transformation .jsp file opens in the Page Designer along with the Insert Host Component wizard.
6. Click **Cancel** on the Insert Host Component wizard.
7. From the Palette view, under Dojo Layout Widgets, select **TabContainer** and drop it to the transformation .jsp design area.
8. In the Insert Tab Container dialog, specify 2 for the **Number of tabs** and click **OK**.
9. In the Properties view for the tabs, or the Source view, change the titles of the tabs. Change `Tab1` to `Table` and `Tab2` to `Graph`.
10. From the Palette view, under HATS Components, select **Table (visual)** and drop it to the Table pane.
11. In the Insert Host Component wizard, on the Screen Region page, select the screen region to be displayed as a visual table. For this example, select the region including the four columns of data and the dashed lines above the first row of data.
12. On the Rendering Options page, select **Table (visual)** from the Components list and click the **Component Settings** button.
13. On the Settings - Table (visual) page, clear the **Use project defaults** box, set **Rows to exclude** to 1, select **Extract column header test from row above table** and click **OK**.
14. On the Rendering Options page, select **Enhanced grid (Dojo)** from the list of widgets and click **Finish**.

15. Press `Ctrl-S` to save your work so far.
16. From the Palette view, under HATS Components, select **Table (visual)** and drop it to the Graph pane.

    Note: If you have difficulty switching to the Graph pane in the Design view, drop the Table (visual) component to before the `</div>` tag for the Graph pane in the Source view.
17. In the Insert Host Component wizard, on the Screen Region page, select the same region you selected for the Table pane.
18. On the Rendering Options page, select **Table (visual)** from the Components list and click the **Component Settings** button.
19. On the Settings - Table (visual) page, clear the **Use project defaults** box, set **Rows to exclude** to 1, select **Extract column header test from row above table** and click **OK**.
20. On the Rendering Options page, select **Graph (horizontal bar)** from the list of widgets and click the **Widget Settings** button.
21. On the Settings - Graph (horizontal bar) page:
    a. Clear the **Use project defaults** box.
    b. Set the **Number of data sets** to 1.
    c. Type `Inventory` for the **X-axis title**.
    d. Type `Part Name` for the **Y-axis title**.
    e. Under **Extract data point labels**, set **Column** to 2.
    f. Click **Data sets**.
22. On the Data Source Settings page, set **Data set 1, column** to 3, select **Orange** in the **Color** drop-down, delete the **legend label**, and click **OK**.
23. On the Settings - Graph (horizontal bar) page, click **OK**.
24. On the Rendering Options page, click **Finish**.
25. Press `Ctrl-S` to save your work.
26. In the Page Designer, click the **Preview** tab to see a preview of your transformation.
27. Run the project. You can adjust the height and width of the TabContainer widget to get the desired display at runtime. Note that you must use fixed height and width on the TabContainer widget in case the tabbed folder is not displayed at runtime.

Below is the source for the table area containing the TabContainer in this example.

```
<!-- Insert your HATS component tags here. -->

<table width="783" height="500" cellspacing="0" cellpadding="0" border="0">
   <!-- flm:table -->
   <tbody>
         <tr>
            <td height="31" width="12"></td>
            <td width="771"></td>
         </tr>
         <tr>
            <td height="469"></td>
            <!-- flm:cell -->
            <td valign="top">
            <div id="TabContainer" dojoType="dijit.layout.TabContainer"
                 tabposition="top" style="height: 469px; width: 771px">
            <div dojoType="dijit.layout.ContentPane" title="Table" id="Tab1">
                 <HATS:Component
                     row="7" erow="16" col="14" ecol="77"
                       alternate="" alternateRenderingSet="" textReplacement=""
                     widget="com.ibm.hats.transform.widgets.dojo.EnhancedGridWidget"
                      widgetSettings=""
                     type="com.ibm.hats.transform.components.VisualTableComponent"
                      componentSettings="minRows:1|validateCharacterType:false|
```

```
                includePreviousLineAsHeader:true|columnBreaks:|minColumns:1|
                behaveAsInDefaultRendering:false|areaAsInDefaultRendering:false|
                overrideRecognitionBehavior:false|evaluateBetterTableParameter:numberOfCells|
                excludeCols:|numberOfTitleRows:0|columnDelimiter: |
                betterTableHasLeastOfParameter:false|includeEmptyRows:true|
                excludeRows:1|" />
        </div>
        <div dojoType="dijit.layout.ContentPane" title="Graph" id="Tab2">
            <HATS:Component
                row="7" erow="16" col="14" ecol="77"
                  alternate="" alternateRenderingSet="" textReplacement=""
                widget="com.ibm.hats.transform.widgets.HorizontalBarGraphWidget"
                 widgetSettings="extractSource:col|backgroundImage:|dataSource1Color:#ff8040|
                  backgroundColor:#ffffff|extractLabels:true|dataSource3Legend:Series 3|
                  dataSource2Color:#00ff00|yAxisTitle:Part Name|textAntialiasing:true|
                  labelIndex:2|extractDataSetLabels:false|height:400|alternateText:Graph.jpg|
                  width:400|dataSource3:3|defaultFont:SansSerif-PLAIN-12|dataSource2:2|
                  dataSource1:3|dataSetNumber:1|dataSource2Legend:Series 2|
                  dataSource3Color:#ff0000|dataSource1Legend:|xAxisTitle:Inventory|
                  axisColor:#000000|labelColor:#000000|dataSetLabelIndex:1|"
                type="com.ibm.hats.transform.components.VisualTableComponent"
                 componentSettings="minRows:1|validateCharacterType:false|
                  includePreviousLineAsHeader:true|columnBreaks:|minColumns:1|
                  behaveAsInDefaultRendering:false|areaAsInDefaultRendering:false|
                  overrideRecognitionBehavior:false|evaluateBetterTableParameter:numberOfCells|
                  excludeCols:|numberOfTitleRows:0|columnDelimiter: |
                  betterTableHasLeastOfParameter:false|includeEmptyRows:true|
                  excludeRows:1|" />
        </div>
        </td>
       </tr>
    </tbody>
</table>
```

The following figures show this example TabContainer widget displayed at runtime. On the Table tab notice the data rendered using the HATS Dojo Enhanced grid widget.

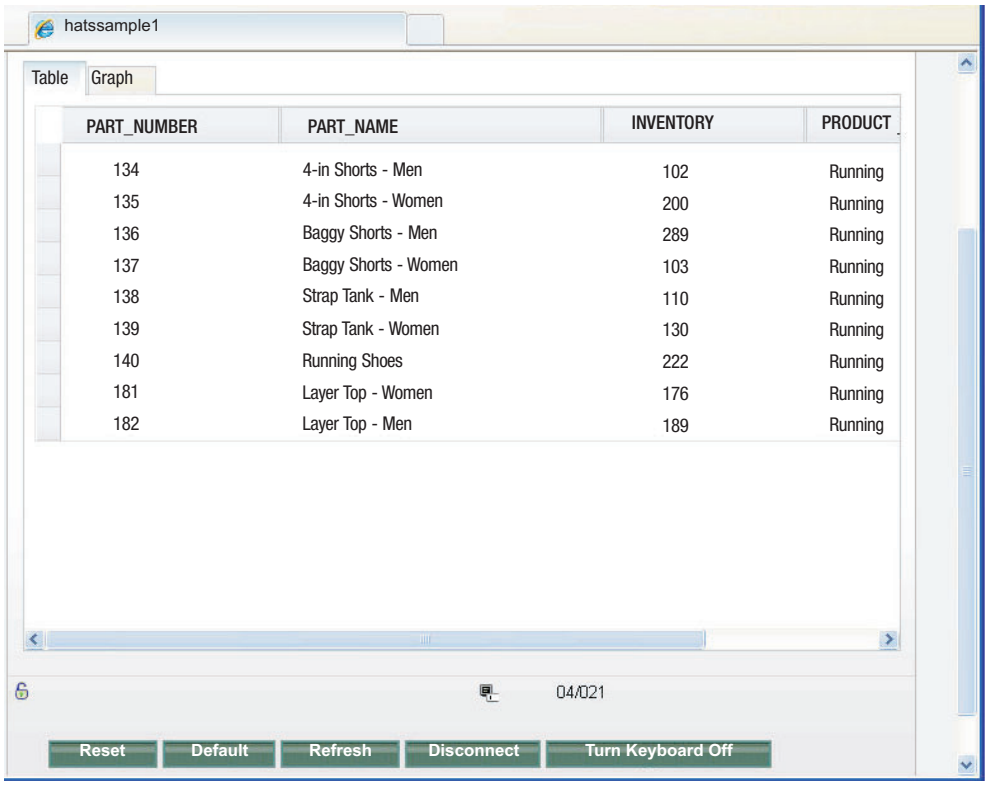


Figure 18. Dojo Enhanced grid widget in Dojo TabContainer widget

On the Graph tab notice the data rendered using the HATS Graph (horizontal bar) widget.

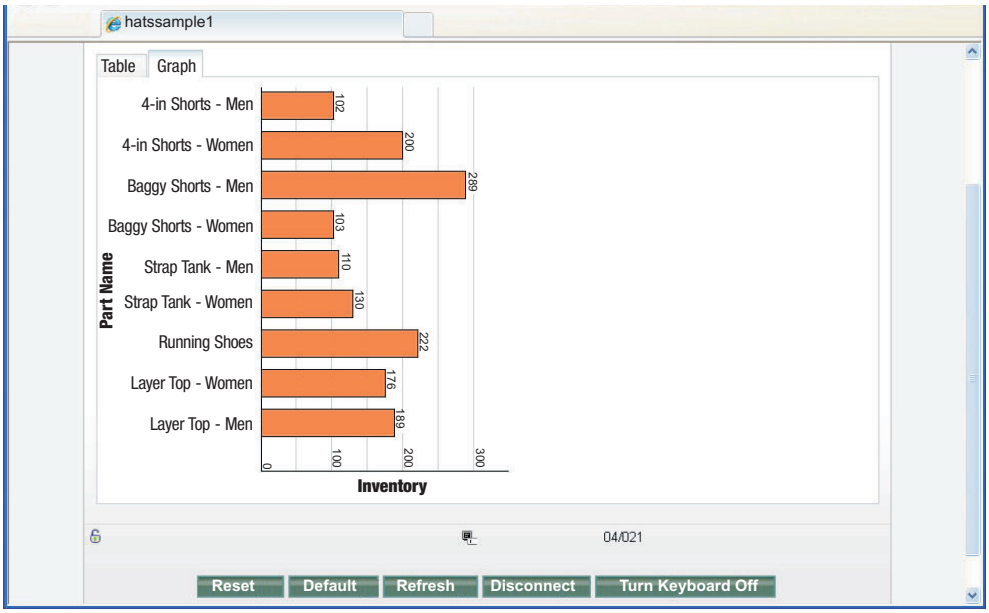


Figure 19. Graph (horizontal bar) widget in Dojo TabContainer widget

Using the Dojo TabContainer widget in a HATS portlet project

You can use the Dojo TabContainer widget in a HATS portlet project.

Note: HATS Dojo widgets are not supported in HATS portlets. When creating tabs using the Dojo TabContainer widget, use HATS non-Dojo widgets to render host components within the tabs.

To use the Dojo TabContainer widget in a HATS portlet project, follow the steps similar to using the widget in a HATS Web project, with the following considerations.

1. The **Use Dojo technology** option is not available when creating a HATS portlet project. After a HATS portlet project is created, in the HATS Projects view right-click on the project, select **Properties -> Project Facets**. Expand the **Web 2.0** project facet and select **Dojo Toolkit on WebSphere Portal 1.0**. This makes Dojo widgets available for selection on the Palette view.
2. When dragging the Dojo TabConatiner into the screen transformation, on the Specify a jsp file for Dojo bootstrap entries page, select **Generate in the portlet jsp file**, clear **Generate portlet helper JavaScript classes in portlet application**, and click **OK**.
3. If dragging the Dojo TabContainer widget to the Design view does not cause the widget to be added to the source, a workaround is to drag the widget to the location after the `<TD>` tag in the Source view.
4. The following TabContainer `<div>` tag is created.

   Note: The user interface for entering the number of tabs is not prompted and the ContentPane is not generated.

```
<div dojoType="dijit.layout.TabContainer"
    id="tabContainer_<portletAPI:namespace/>"
       style="width: 500px; height: 100px"></div>
```

5. To create tabs, drag ContentPane widgets (under Dojo Layout Widgets from the Palette view) to before the tag, resulting in the following source:

```
<div dojoType="dijit.layout.TabContainer"
    id="tabContainer_<portletAPI:namespace/>"
       style="width: 500px; height: 100px">
<div dojoType="dijit.layout.ContentPane"
     id="contentPane_<portletAPI:namespace/>"></div>
<div dojoType="dijit.layout.ContentPane"
     id="contentPane_<portletAPI:namespace/>"></div>
<div dojoType="dijit.layout.ContentPane"
     id="contentPane_<portletAPI:namespace/>"></div>
</div>
```

6. Add titles and insert a HATS component into each ContentPane widget.

   Note: You can see the layout in Design view, but nothing is displayed in the Preview view.

7. Export the portlet and deploy it to a Portal server.

   Note: If necessary, you can adjust the height and width of the TabContainer widget to get the desired display. Note that you must use fixed height and width on the TabContainer widget in case the tabbed folder is not displayed at runtime.

Below is example source for a TabContainer widget, containing three tabs, rendering an IBM i Main Menu screen. The tabs contain the HATS Selection list, Command line, and Function key host components, respectively.

```
<!-- Insert your HATS component tags here. -->

<table width="100%" height="100%" cellspacing="0" cellpadding="0" border="0">
   <!-- flm:table -->
   <tbody>
         <tr>
            <td>
            <div dojoType="dijit.layout.TabContainer"
                id="tabContainer_<portletAPI:namespace/>"
                 style="width: 500px; height: 400px">
            <div dojoType="dijit.layout.ContentPane"
                id="contentPane_<portletAPI:namespace/>" title="MenuOption">
                 <HATS:Component
                       row="5" erow="17" col="1" ecol="80"
                       alternate="" alternateRenderingSet="" textReplacement=""
                      widget="com.ibm.hats.transform.widgets.SLRadioButtonWidget"
                      widgetSettings=""
             type="com.ibm.hats.transform.components.SelectionListComponent"
             componentSettings=""  />
            </div>
            <div dojoType="dijit.layout.ContentPane"
           id="contentPane_<portletAPI:namespace/>" title="CommandLine">
                 <HATS:Component
              row="19" erow="21" col="1" ecol="80"
                       alternate="" alternateRenderingSet="" textReplacement=""
              widget="com.ibm.hats.transform.widgets.InputWidget"
              widgetSettings=""
              type="com.ibm.hats.transform.components.CommandLineComponent"
              componentSettings=""  />
            </div>
            <div dojoType="dijit.layout.ContentPane"
           id="contentPane_<portletAPI:namespace/>" title="FunctionKey">
                 <HATS:Component
                row="22" erow="23" col="1" ecol="80"
                         alternate="" alternateRenderingSet="" textReplacement=""
                widget="com.ibm.hats.transform.widgets.ButtonWidget"
                widgetSettings=""
                type="com.ibm.hats.transform.components.FunctionKeyComponent"
                componentSettings=""  />
```

```
            </div>
            </div>
            </td>
         </tr>
      </tbody>
   </table>
```

Chapter 5. Programming in HATS Portlets

This chapter assumes that you are familiar with developing portlets for WebSphere Portal. The *HATS User's and Administrator's Guide* explains how to create HATS portlets and how to convert HATS Web projects into portlet projects. You must have WebSphere Portal Toolkit installed on your HATS machine. With Portal tools, you can create, manage, deploy, and run portlets in a Rational SDP environment. WebSphere Portal Toolkit is installed with Rational SDP. When installing Rational SDP, select the **Portlet and Portal development tools** check box.

You can develop HATS portlets that comply with the standard Java Portlet Specification API (JSR 168 or JSR 286), hereafter referred to as the standard portlet API. HATS portlets that comply with the standard portlet API (JSR 168 or JSR 286) are referred to as standard portlets. Where a distinction is required, JSR 168 or JSR 286 is noted.

Portlets can be targeted for **WebSphere Portal v7.0** or **WebSphere Portal v8.0**.

The Portal API documentation (Javadoc) is installed with the toolkit and contains information that is useful in performing the tasks that are described in this chapter. You can use the Portal API in business logic or JSPs. A detailed example of using HATS business logic with the Portal API, titled *Portlet messaging with IBM Rational Host Access Transformation Services (HATS)*, can be found on the HATS Knowledge Center.

Standard portlets

The following topics are specific to standard portlets.

Using security

If you are using a credential vault with your WebSphere Portal, you can configure your HATS portlets to work with the credential vault. HATS provides a Web Express Logon plug-in called WebSphere Portal Credential Vault Credential Mapper. This plug-in appears in the Add Credential Mapper plug-in window only for a portlet project. This plug-in retrieves a passive user-password credential from a vault slot.

The following classes are provided to aid the access to the Portal Credential Vault for standard portlets.

- com.ibm.hats.portlet.cv.CredentialVaultHelper
- com.ibm.hats.portlet.cv.UserPasswordCredential

The CredentialVaultHelper class has the following public methods. Note that the getInstance() method returns a unique instance of CredentialVaultHelper.

```
public static CredentialVaultHelper getInstance();
public static String generateSlotName(String vaultId,String hostDestination,String hostAppId);
public String getSlotId(PortletRequest portletRequest, String slotName, int slotType);
public void setCredential(String slotId,UserPasswordCredential credential,PortletRequest portletRequest);
public UserPasswordCredential getCredential(String slotId,PortletRequest portletRequest);
```

The UserPasswordCredential class has the following public methods.

```
public UserPasswordCredential();
public UserPasswordCredential(String user,char[] password);
public void setUser(String user);
```

```
public void setPassword(char[] password);
public void setPassword(String password);
public String getUser();
public String getPassword();
```

Refer to the HATS API References (Javadoc) in the HATS Knowledge Center at http://www.ibm.com/support/knowledgecenter/SSXKAY_9.5.0 for detailed information about the com.ibm.hats.portlet.cv.CredentialVaultHelper and com.ibm.hats.portlet.cv.UserPasswordCredential classes.

You are responsible for creating and populating the vault slot for your users. The Web Express Logon plug-in can be used directly with credentials created using the setCredential() method of the com.ibm.hats.portlet.cv.CredentialVaultHelper class, because it observes the same naming convention for the slots. The vault slot name can be generated using the generateSlotName() method of the com.ibm.hats.portlet.cv.CredentialVaultHelper class, where you pass in the plug-in parameter SLOT_ID, the host name, and the application name (use null if the application name is not applicable, for example, when connecting to an IBM i server). Note that the generated slot name is the SLOT_ID concatenated using spaces with the host name, then the application name. The three elements of the slot name are encoded to replace spaces with underscores. The actual vault slot ID can then be retrieved using the getSlotId() method of the com.ibm.hats.portlet.cv.CredentialVaultHelper class, where you pass in the PortletRequest, the slot name, and the SLOT_TYPE. Note that if the SLOT_TYPE is 2 or 3, the slot ID is equal to the slot name.

You can populate the vault slot with credentials that are specified in your business logic or retrieved from another source. If you want to use the Portal user ID, you can retrieve it using the WebSphere Portal Network Security plug-in. This plug-in appears in the Add Network Security plug-in window only for a portlet project.

As described in Chapter 10, “Creating plug-ins for Web Express Logon,” on page 103, Web Express Logon uses two types of plug-ins, Network Security plug-ins and Credential Mapper plug-ins. Any of the Network Security plug-ins that are supplied with HATS can be used in a HATS portlet. Table 3 lists some possible combinations of plug-ins you can use in your portlet:

Table 3. Plug-in combinations

Network Security plug-in	**Credential Mapper plug-in**
WebSphere Portal NS plug-in	Any supplied or custom CM plug-in or WebSphere Portal Credential Vault CM plug-in
None	WebSphere Portal Credential Vault CM plug-in
Custom NS plug-in	Custom CM plug-in

The following steps are an example of how you can add Web Express Logon capability to your portlet. Add this logic to the Start event for your HATS portlet. When a user opens the portlet, check a global variable to determine whether the user’s host credentials have already been supplied.

1. If the host credentials have already been supplied, show the user the first screen to be displayed after authentication. This might be the screen that appears at the end of the Web Express Logon logon macro.

2. If the host credentials have not already been supplied, use the getCredential() method of the com.ibm.hats.portlet.cv.CredentialVaultHelper class to request the user's credentials from the vault.
   a. If the credentials are received successfully (the method does not return null), do the following:
      1) Set the global variable to show that the user's host credentials have been supplied.
      2) Run the Web Express Logon logon macro.
   b. If getCredential() returns null, do the following:
      1) Present a sign-on screen to request the user's information. This can be a transformed host screen or an HTML page you have created for this purpose.
      2) Store the input in global variables and add it to the vault.

Extending the Entry portlet

You can extend the HATS standard portlet to provide additional functions or to customize the portlet. For example, you can extend the portlet to provide connection parameter and global variable overrides.

To extend the standard Entry portlet you must customize the Java class file. The following steps are for extending the JSR 168 class file (com.ibm.hats.portlet.Jsr168EntryPortlet). The same steps can be performed to extend the JSR 286 class file (com.ibm.hats.portlet.Jsr286EntryPortlet).

1. Use the new Java Class wizard to create the new Java class that extends com.ibm.hats.portlet.Jsr168EntryPortlet.
   a. Click **File > New > Java > Class file**.
   b. Provide the package and the name of the new class. Make sure the **Superclass** field contains the value `com.ibm.hats.portlet.Jsr168EntryPortlet`.
2. Modify the name of the portlet class in the portlet deployment descriptor.
   a. Open the portlet.xml file in the Portlet deployment descriptor editor.
   b. Replace the text in the <portlet-class>`com.ibm.hats.portlet.Jsr168EntryPortlet`</portlet-class> with the following: <portlet-class>`myPackage.myClass`</portlet-class>; where myPackage and myClass are values you entered in the new Java Class wizard above.

The following example shows how to provide connection parameter and global variable overrides with the extended portlet class:

1. Edit the Java source file of the extended portlet class.
2. Override the getInitParameters() method as follows:

```
public Properties getInitParameters(IRequest request)
{
  Properties initParams = new Properties();
  initParams.setProperty(com.ibm.eNetwork.beans.HOD.Session.HOST, "129.12.11.2");
  initParams.setProperty(com.ibm.eNetwork.beans.HOD.Session.PORT, "623");
  initParams.setProperty("hatsgv_userName", "some user");
  initParams.setProperty("hatsgv_password", "xxxxx");
  initParams.setProperty("hatssharedgv_someVariable", "zzzzz");
  return initParams;
}
```

Note: If you set the connection parameter or global variable overrides through the extended portlet class and use the same overrides in Edit mode, the Edit mode settings override the settings in the extended portlet class.

You must also configure the project's security settings to allow overrides of your chosen connection parameters or global variables. To do this, use the Connection Parameter Overrides panel and the Global Variable Overrides panel in the **Project Settings>Other** tab. If you do not allow overrides in the security panels, then any overrides set by extending the portlet class or by using the portlet Edit mode are implicitly ignored.

Running Integration Objects

The processRequest() method is used to run Integration Objects in standard portlets and special considerations have to be made for Integration Object chaining.

For Integration Object chaining (see "Integration Object chaining" on page 59) the same connection must be used by all Integration Objects in the chain. When using the processRequest() method to run the Integration Object, the key that represents the connection for the Integration Object chain must be extracted from the first Integration Object in the chain and set on subsequent Integration Objects in the chain. The connection can be extracted from the first Integration Object using the getHPubLinkKey() method and set on subsequent IOs using the setHPubLinkKey().

The key representing the connection is passed across JSP as a parameter on the HttpServletRequest object by adding a hidden input field to a FORM as shown:

```
<INPUT TYPE="HIDDEN" NAME="<%CommonConstants.HPUB_LINK_KEY%>"VALUE="<%=FirstInChainIO.getHPubLinkKey()%>"/>
```

The subsequent JSP then retrieves it using the getParameter() method and sets it on the subsequent Integration Object as shown:

```
<% MiddleInChainIO.setHPubLinkKey((String)request.getParameter(CommonConstants.HPUB_LINK_KEY)); %>
```

When the Forward To URL action is used to pass the connection being used by the HATS Portlet to the Integration Object(s), the key representing the connection is saved in the CommonConstants.HPUB_LINK_KEY attribute of the PortletRequest object.

In this case, the user must edit the JSP that gets control from the Forward To URL Action to retrieve the key from the PortletRequest object and set it for the first Integration Object calling the setHpubLinkKey() method, as shown below:

```
<% ExampleIO.setHPubLinkKey((String)request.getAttribute(CommonConstants.HPUB_LINK_KEY)); %>
```

The following statements also have to be added to the JSP that gets control from the Forward To URL action, regardless of whether it's an input page that gathers the data required by the Integration Object, or an output page that presents the results after the Integration Object is run:

```
<%@ taglib uri="http://java.sun.com/portlet" prefix="portletAPI" %>
<portletAPI:defineObjects/>
```

If the JSP that gets control from the Forward To URL action is an input page, the application developer is also required to modify the FORM statement by replacing the use of the request and response objects with the portlet renderRequest and renderResponse objects as shown below:

```
<FORM NAME="iojsp_ExampleIOOutput" METHOD="POST"
ACTION='<%= renderResponse.encodeURL(renderRequest.getContextPath() + "/iojsp/ExampleIOOutput.jsp")%>'>
```

Using Web Express Logon

To run an Integration Object that is configured to use Web Express Logon in standard portlets, you must ensure PortletRequest is available in the Integration Object by calling setHPubPortletRequest(javax.portlet.PortletRequest) before executing the processRequest() method. For example, to run the Integration Object in a JSP page where the portlet renderRequest is available with the following statements:

```
<%@ taglib uri="http://java.sun.com/portlet" prefix="portletAPI" %>
<portletAPI:defineObjects/>
```

add the following statement:

```
<% ExampleIO.setHPubPortletRequest(renderRequest); %>
```

before the statement:

```
<% ExampleIO.processRequest(); %>
```

Adding JavaServer Pages to a portlet

If you copy a .jsp file from a non-portlet project into a HATS portlet project, you must convert it by right-clicking the file name and selecting **Convert JSP for Portal**. You can convert several .jsp files at a time by selecting them in the tree view. This menu item is available only if the Portal Toolkit has been installed and the .jsp is in a HATS portlet project and has not already been converted. If you select several .jsp files and one or more of the files have already been converted, they will be skipped.

Chapter 6. Programming with Integration Objects

Integration Objects are Java beans that encapsulate interactions with a host application. If you have used IBM WebSphere Host Publisher, you are already familiar with most aspects of Integration Objects, but you will need to learn about how HATS enables you to work with Integration Objects. You do not need to be familiar with Host Publisher to use Integration Objects in HATS.

HATS User's and Administrator's Guide explains how to create an Integration Object from a macro and how to create Web pages based on an Integration Object. Integration Objects created using IBM WebSphere Host Publisher can be imported into HATS and used in the same ways as Integration Objects that were created in HATS Toolkit. This document describes these advanced uses of Integration Objects:

- Invoke an Integration Object from business logic. This process is described in "Example: Calling an Integration Object" on page 10.
- Invoke an Integration Object from another WebSphere application. This process is described in "Using Integration Objects in a WebSphere Java EE application" on page 96.
- Create Web services based on one or several Integration Objects. This process is described in Chapter 7, "Developing Web services," on page 65.
- Create EJB Access Beans based on one or several Integration Objects. This process is described in Chapter 8, "Creating and using a HATS EJB application," on page 81.
- Modify the Java code contained in the Integration Object. This is described in Chapter 9, "Integration Objects - advanced topics," on page 91.

A common class for accessing Integration Object information

The properties of an Integration Object can be accessed from WebSphere applications. The calling program must know the name of the Integration Object and the name of the variable. Sometimes, it is advantageous for the calling program to be able to access properties that all Integration Objects share without knowing the name of the Integration Object. The Java class, com.ibm.HostPublisher.IntegrationObject.HPubCommon, is extended by all Integration Objects. The class com.ibm.HostPublisher.IntegrationObject.HPubHostAccess extends HPubCommon and is common to all Host Access Integration Objects. The class com.ibm.HostPublisher.IntegrationObject.HPubDBAccess extends HPubCommon and is common to all Database Integration Objects. You can extract information from these classes without knowing the name of the Integration Object. Introspect these classes to find the names of the current properties that can be extracted using the Integration Object methods. For information about these methods, see "Integration Object methods" on page 54.

Java class hierarchy of Integration Objects

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

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

Integration Object methods

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

Common methods

These methods are common to all Integration Objects.

void doHPTransaction(HttpServletRequest req, HttpServletResponse resp) throws BeanException

This execution method runs a HATS Integration Object or EJB Access Bean from a servlet or JSP. If you use this method, HATS manages Integration Object chaining. However, you must ensure that the Integration Objects in your Web project are chained together in the correct order.

void processRequest() throws BeanException

This execution method runs an Integration Object or EJB Access Bean in environments where there is no HttpServletRequest or HttpServletResponse (environments other than a Web module). To run chained Integration Objects using this method, additional programming is required; refer to "Integration Object chaining" on page 59.

Note: To run an Integration Object that is configured to use Web Express Logon in standard portlets, you must ensure PortletRequest is available in the Integration Object by calling setHPubPortletRequest(javax.portlet.PortletRequest) before executing the processRequest() method.

java.lang.String getHPubBeanName()

Returns the name of the current Integration Object or EJB Access Bean.

java.lang.String getHPubBeanType()

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

HOD The bean was created using Host Access.

DB This bean was created using Database Access.

void setHPubErrorPage(java.lang.String value)

For Integration Objects that were created with Host Publisher Version 2.2.1 or Version 3.5, this method sets the name of the error page to be used. Use this method only if you are running the HATS Integration Object or EJB Access Bean from a servlet or JSP. Specify the name of your error page relative to the location of your servlet or JSP.

Note: This method is deprecated and cannot be used for Integration Objects that were created with HATS or with Host Publisher after Version 3.5.

java.lang.String getHPubStartPoolName()
Returns the name of the connection pool from which the Integration Object acquired the connection.

void setHPubStartPoolName(java.lang.String value)
Sets the name of the connection pool from which the Integration Object will acquire the connection. If the `processRequest()` business method of the Integration Object is being used (for example, when the Integration Object is deployed in an EJB container or as a Web service), the pool name must be qualified with the HATS application name. For example, the pool name should be *my_hats_project*/main.

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

java.lang.String getHPubXMLProperties(HPubConvertToTableFormat.xsl)
Returns an XML formatted string that specifies the property names and values for this Integration Object, and applies XML style sheet processing to the returned string. See "Applying XML style sheet processing to Integration Object output" on page 61 for more information.

void setHPubSaveConnOnError(java.lang.Boolean flag)
Sets an indicator in the Integration Object that specifies that the connection should not be destroyed if an error is detected while executing the Integration Object. Instead, the connection should be saved so that it can be passed to the HATS entry servlet and a default transformation can be applied. The method should be used in combination with the predefined AdvancedIOErrorPage.jsp. The connection can be transformed only if it was obtained from the default connection pool. This method cannot be used with EJB Access Beans or HATS Web services support.

int getHPubErrorOccurred()
Returns a nonzero value when an error has occurred.

java.lang.Exception getHPubErrorException()
Returns an exception object that describes the error that occurred; valid only if HPubErrorOccurred is nonzero. This property is not contained in the *io_name*_Output_Properties class that is generated by HATS Web services support because parameters of type java.lang.Exception cannot be serialized over Simple Object Access Protocol (SOAP.)

java.lang.String getHPubErrorMessage()
Returns a string containing the HATS code and message of the error that occurred; valid only if HPubErrorOccurred is nonzero.

Host Access Integration Object methods

These methods can be used in Host Integration Objects created with Host Publisher and imported into HATS, and with Integration Objects created in HATS Toolkit. They cannot be used with Database Access Integration Objects created with Host Publisher.

java.lang.String getHPubLinkKey()
This method returns the name of the key that represents the connection for

the Integration Object chain. This value should be obtained from the first Integration Object in a chain after the Integration Object has run in a non-Web container.

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

java.lang.String getHPubStartChainName()
This method returns the name of the start state label as defined when a middle or last in chain Integration Object is created. This value is `Null` for the first Integration Object in a chain or an Integration Object that is not chained.

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

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

java.lang.String getHPubMacroMessage()
This method returns the value of the message tag of the last screen that was executed in the current Host On-Demand macro screen.

public java.lang.String getHPubConnectionOverrides()
This method returns the connection overrides used by IO in the format "key1=value1, key2=value2" or an empty string.

See Specifying Connection Overrides for further information.

public void setHPUBConnectionOverrides (String overrides)
This method specifies the connection overrides to apply when establishing the host connection for the Integration Object. The connection overrides have to be set before calling the processRequest() or doHPTransaction() methods of the Integration Object. The connection overrides must be in the format "key1=value1, key2=value2". This format is more convenient when building Web services.

This method should be used with client programs based on HATS EJB access beans or Web Services clients generated to be used with HATS web services.

If connection overrides have been specified, then when the Integration Object's doHPTransaction() or processRequest() method is called a new connection pool for the Integration Object is created. The new pool is based on the Integration Object's original connection pool and the supplied connection overrides. The Integration Object is automatically switched to use the new connection pool. new pool name can be retrieved using the Integration Object instance method getHPubStartPoolName().

See Specifying Connection Overrides for further information.

public void setHPubConnectionOverrides(Properties overrides)
This method sets the connection overrides to apply when establishing the host connection for the Integration Object. The connection overrides have to be set before calling the processRequest() or doHPTransaction() business methods of the Integration Object establishing the host connection.

When the Integration Object doHPTransaction() or processRequest() methods are called, if there are connection overrides associated with the Integration Object, a new connection pool for the Integration Object is created based on the Integration Object's connection pool and the connection overrides. The new pool name can be retrieved using the Integration Object instance method getHPubStartPoolName().

See Specifying Connection Overrides for further information.

Database Access Integration Object methods

These methods can be used in Database Access Integration Objects that you created with Host Publisher and imported into HATS. They cannot be used with Host Access Integration Objects created with Host Publisher, and they cannot be used with Integration Objects that were created in HATS Toolkit. Refer to the Rational SDP documentation for information about accessing databases.

java.lang.String getHPubWarningOccurred()
Returns a nonzero value that indicates that a warning has occurred.

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

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

Specifying Connection Overrides

Connection overrides can be set dynamically or by the user. Specifying connection overrides allows the user to build a generic application and customize some settings based on the user running the application. However, for example, the LUName being used for a 3270E connection or the workstationID for a 5250 connection can be set dynamically at runtime.

To specify connection overrides on Integration Objects, the user needs to modify the code accessing the Integration Object to call the Integration Object instance methods setHPubConnectionOverrides(Properties overrides) or setHPubConnectionOverrides(String overrides) before calling the doHPTransaction() or processRequest() methods. The method sets the connection overrides for the Integration Object. When the Integration Object doHPTransaction() or processRequest() methods are called, if there are connection overrides associated with the Integration Object, a new pool for the Integration Object is created based on the Integration Object's connection pool and connection overrides in input.

The following is an example of JSP code using the IO.setHPubConnectionOverrides() method:

```
<% // Set the connection overrides
java.util.Properties overrides = new java.util.Properties();
overrides.setProperty("LUName", "LU00001");
// Apply overrides to the IO
SignOn.setHPubConnectionOverrides(overrides);
SignOn.doHPTransaction(request,response);
// Get new IO pool name to be used in following logic
String newPoolName = SignOn.getHPubStartPoolName();

%>
```

To specify connection overrides on existing Integration Objects, without recompiling the Integration Object, the new com.ibm.HostPublisher.IntegrationObject.HPubPoolFactory class can be used. Its static create() method can be used before calling the doHPTransaction() or processRequest() methods. The method generates a new pool object by cloning the pool object in input, and applying the connection overrides to the Host On-Demand properties associated with it. The new pool name is returned and must be set in the Integration Object using the setHPubStartPoolName() method. The create() method returns null, if the pool name in input is invalid.

If the third parameter of the create method is null, then the pool name must be qualified with the HATS application name. For example, the pool name should be my_hats_project/main.

The two static methods that you can use are:

```
static String create(String poolName,
                     Properties overrides,
                     javax.servlet.http.HttpServletRequest httpServletRequest)
static String create(String poolName,
                     String overrides,
                     javax.servlet.http.HttpServletRequest httpServletRequest)
```

The methods create a new pool definition based on the named pool definition and the supplied connection overrides.

The new pool name must be set on the Integration Object using the instance method setHPubStartPoolName() before calling the doHPTransaction() or processRequest() methods of the Integration Object establishing the host connection.

The create() methods returns null if a connection definition for the supplied pool name does not exist in the project.

If the third parameter of the create method is null, then the pool name must be qualified with the HATS application name. For example, the pool name should be "my_hats_project/main".

The connection overrides must be in the format "key1=value1, key2=value2" if the second method signature is used. The first method uses a standard Java Properties object to specify the overrides.

The following is an example of JSP code using the com.ibm.HostPublisher.IntegrationObject.HPubPoolFactory,create() method:

```
<% // Set the connection overrides
java.util.Properties overrides = new java.util.Properties();
overrides.setProperty("LUName", "LU00001");
// Create a new Pool based on a the default pool and the connection overrides
String poolName = SignOn.getHPubStartPoolName();
String newPoolName =
com.ibm.HostPublisher.IntegrationObject.HPubPoolFactory.create
                                (poolName, overrides, request);
// If a valid poolName is returned, make the IO use the new pool
if (newPoolName != null) {
  SignOn.setHPubStartPoolName(newPoolName);
  SignOn.doHPTransaction(request,response);
}
else {
```

```
  // Error condition
}

%>
```

For chained Integration Objects to work correctly:

- If you use the setHPubConnectionOverrides() method, the connection overrides must be set on the first Integration Object in the chain.
- If you are using the HPubPoolFactory.create() method, the newly created pool name must be set on the first Integration Object in the chain.

With either example, the Integration Object uses a new Pool object created by cloning the original Integration Object's pool and applying the connection overrides. All the settings that apply to the original Integration Object's pool also apply to the new pool, including pooling. Connection overrides can be different for each user accessing an Integration Object. HATS runtime creates a pool object for each user. These pools, created dynamically when connection overrides are specified, are destroyed when the last active connection is terminated, providing that pooling is not enabled. Connection override parameters specified on the Connection Parameter Overrides page on the Other tab in the project settings editor do not apply to Integration Objects.

Pools dynamically created when connection overrides are specified are automatically destroyed when the last active connection is terminated, providing that pooling is not enabled.

Integration Object chaining

Integration Object chaining can break up a complex application into multiple tasks, with each task represented by an Integration Object. Chaining enables you to run several Integration Objects in sequence, with each Integration Object depending on the one before it for its input. Refer to *HATS User's and Administrator's Guide* for an introduction to Integration Object chaining. Integration Object chaining is quite different from macro chaining in that each Integration Object in a chain is run to completion before the next Integration Object in the chain takes control. Macro chaining, using the PlayMacro action, terminates the current macro (the one in which the PlayMacro action occurs) and begins to process the specified macro screen of the target macro. There is no return to the calling macro. See *HATS Advanced Macro Guide* for more information on macro chaining.

Integration Object chaining is handled by HATS for the following:

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

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

Properties that enable chaining must be retrieved and set for the following:

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

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

See "Integration Object methods" on page 54 for a description of the `doHPTransaction` and `processRequest` methods.

HATS provides methods that enable you to extract the key that represents the connection for the Integration Object chain from the first Integration Object in a chain and to set the property for subsequent Integration Objects in the chain. Properties that enable chaining for Web Services must also be retrieved and set.

To build an Integration Object chain using the `processRequest` method, do the following:

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

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

   where *Xyz* is the name of your input variable.
3. Invoke the Integration Object to perform its task (running a macro, for example), using the method:

   ```
   IOChain1.processRequest()
   ```
4. Check for errors by invoking:

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

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

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

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

   where *Xyz* is the name of your input variable.
9. Invoke this Integration Object to perform its task, using the method:

   ```
   IOChain2.processRequest()
   ```
10. Check for errors by invoking:

    ```
    IOChain2.getHPubErrorOccurred()
    ```

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

HATS-chained Web services

HATS-chained Web services require special consideration when used with Integration Objects. If you use chained Integration Objects within a HATS Web service, you create a stateful Web service. HATS runtime does not store data between invocations of chained Integration Objects in the same Web service. However, HATS runtime does require that the next in chain Web service invocation be routed back to the same instance of the HATS runtime, so the next-in-chain Integration Object uses the same Telnet connection.

If you are deploying the stateful HATS Web service to a cluster, you can ensure that this occurs in one of the following ways:

1. Create HATS Web services from EJB Access Beans. The HATS EJB is a stateful session EJB, so the EJB client always interacts with the same HATS EJB instance.
2. Configure the scope of the Web service to be session (either in the development environment or after deployment), and use the SESSION_MAINTAIN_PROPERTY in the Web service client runtime to maintain the session across invocations, ensuring HTTP session affinity.

Applying XML style sheet processing to Integration Object output

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

For information on the methods that you can use in WebSphere applications, see "Integration Object methods" on page 54.

DTD of XML data that is returned by getHPubXMLProperties() method

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

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

XML data using the getHPubXMLProperties() method

The following example shows sample data and the resulting XML data.

Table 4. Sample XML data

Name	Phone Number
Mary Smith	765–4321
John Doe	123–4567

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

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

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

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

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

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

XML data with HPubConvertToTableFormat style sheet applied

The sample data shown in Table 4 on page 61 results in the following XML data:

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

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

```
</DataRecord>
</Table>

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

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

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

Chapter 7. Developing Web services

Web services are self-contained applications, based on open standards, that can be invoked over the Web. Web services provide a way for applications to connect and interact easily and efficiently. They can be building blocks of applications used within your enterprise or provide a point of interaction with other enterprises. Because their interfaces are defined according to standards, Web services can interact with other applications that are not Java-based.

You can use HATS Web service support to create service-oriented architecture (SOA) assets that provide standard programming interfaces to business logic and transactions contained within your host applications. These core business tasks can be reused as standard Web services that participate as an integral part of your business process integration plan. Use your core business services as building blocks to develop new internal applications or to integrate with applications outside your enterprise. Your host-based business tasks can then be included in your SOA solutions with IBM SOA Foundation products, such as IBM WebSphere Process Server, IBM WebSphere Enterprise Service Bus, and others.

Using a combination of tools from HATS Toolkit and Rational SDP, you can create Web services from Integration Objects or from EJB Access Beans. With HATS you can create traditional Web services defined by Web Services Description Language (WSDL) files as well as Representational State Transfer (RESTful) Web services.

Note: You cannot use an Integration Object that is configured to use Web Express Logon in a Web service.

Traditional Web services use several standard ways to formulate information: Simple Object Access Protocol (SOAP), Web Service Description Language (WSDL), Universal Description, Discovery, and Integration (UDDI). This book refers to but does not explain these protocols. You can find information about them by opening the Rational SDP documentation (click **Help > Help Contents** from any Rational SDP perspective) and searching for **Web service**.

RESTful Web services provide an alternative to the traditional WSDL-style Web service implementation and may be more appropriate for your particular needs. RESTful Web services use HTTP instead of SOAP, and may require less bandwidth, which may be useful for devices like mobile phones and PDAs. In addition, use of the HTTP caching infrastructure (with the HTTP GET method) may improve performance for data that can be cached.

Note: Traditional Web services may continue to be more appropriate for cases where a formal description (the WSDL file) of the Web service interface must be established.

RESTful Web services use a stateless architecture and are viewed as resources rather than function calls. They use well-formatted URIs to identify resources, use HTTP method protocols to do create, retrieve, update, and delete (CRUD) activities, and use HTTP header information to define the message formats. For more information about RESTful Web services, see the Rational Business Developer Knowledge Center at http://www.ibm.com/support/knowledgecenter/SSMQ79 and search for **Architectural styles in web services**.

Usually you create HATS Web services from Integration Objects that you have already built, tested, and deployed. This chapter assumes that you have one or more successfully tested Integration Objects that you want to include in a Web service. However, you can start from the very beginning by opening your host terminal, recording one or more macros, creating Integration Objects from the macros, and testing the Integration Objects. Refer to *HATS User's and Administrator's Guide* for information about creating an Integration Object from a macro.

Creating traditional (WSDL-based) Web services

You must decide whether you want your Web service to comply with the Web Services Interoperability (WSI) standard. This will govern which Web service runtime you use for the creation of your Web services. If you accept the default, IBM WebSphere runtime, the resulting Web service will be WSI-compliant.

You must follow certain naming conventions when creating Web services using IBM WebSphere runtime. These include:

- Method names must begin with a lower case letter.
- Class names must begin with an upper case letter.
- If a method or class name contains an underscore followed by a letter, the letter must be an upper case letter.
- If a method or class name contains a number followed by a letter, the letter must be an upper case letter.

Keep these naming restrictions in mind when creating macro names, prompt names, and extract names during macro creation. For more information on naming restrictions, refer to the Rational SDP documentation.

Creating a Bottom-up Web service from Integration Objects

The instructions in this section assume that you want to create a Web service that has input and output properties and methods similar to all of the properties and methods in an Integration Object. Or, in other words, you want the signature of the Web service that you create to be similar to that of an Integration Object contained in the Web service. This scenario is called a Bottom-up Web service.

To create a Web service, begin from the HATS Projects view and follow these steps:

1. Expand the project that contains the Integration Objects you wish to use, then expand the **Source** folder and the **IntegrationObject** folder.
2. Right-click any Integration Object in your project and select **Create Web Service Support Files**.
3. The Create Web Service Support Files wizard enables you to select any project as the source of the Integration Objects you will include in a Web service. The project in which you clicked will be the default. Provide a class name for the HATS Web service support files. The class name must begin with an uppercase letter. This class is referred to as your *wrapper class* for this Web service. This wrapper class enables you to select a logical group of Integration Objects or EJB Access Beans to include in one Web service. For example, you might select to include all the Integration Objects in a chain in the wrapper class. To see a list of Integration Objects and select the ones to include, click **Next**.
4. Select the resources (Integration Objects and EJB access beans) that you want to include in your Web service. If you want to change the input and output properties that are exposed by the Web service for a given resource, select the resource and click **Properties**.

5. On the Choose Properties page, select the input and output properties that you want exposed in the Web service. If you wish, you can provide an alias name for the property in the Alias Name field. Use the **Select All**, **Deselect All**, and **Select Default** buttons to help in selecting the properties. Click **OK**.

   Notes:

   a. Defaults for single Integration Objects and EJB access beans are all macro prompts for input properties and all macro extracts for output properties.

   b. Defaults for chained Integration Objects are all macro prompts plus hPubLinkKey (which is required) for input properties and all macro extracts plus hPubLinkKey (which is required) for output properties.

   c. Defaults for chained EJB access beans are all macro prompts plus hPubLinkKey and hPubAccessHandle (both of which are required) for input properties and all macro extracts plus hPubLinkKey and hPubAccessHandle (both of which are required) for output properties.

   d. To specify that Integration Object connection overrides be exposed to the Web service, you must select the appropriate properties, for example the hPubConnectionOverrides or hPubStartPoolName properties, which are not selected by default. For more information, see "Specifying Connection Overrides" on page 57.

   e. For more information about the use of other Integration Object methods, see Chapter 6, "Programming with Integration Objects," on page 53.

   f. For considerations when using bidirectional language support, see Support of bottom-up Web services in the *HATS User's and Administrator's Guide*.

6. Click **Finish**. HATS creates a set of classes to be used in creating the Web service. The following classes appear in the HATS Projects view in the Source\webserviceclasses folder:

   - The wrapper class that you specified. This wrapper class contains an *io_name*ProcessWS() method for each Integration Object that you chose to include in the wrapper class. When you create the Web service using Rational SDP wizards, you are creating a Web service that contains all of the *io_name*ProcessWS() methods that are contained in the wrapper class.
   - Input properties classes (*io_name*_Input_Properties) for each Integration Object that you included in the wrapper class. The input properties class is used to set all the necessary inputs for the Integration Object.
   - Output properties classes (*io_name*_Output_Properties) for each Integration Object that you included in the wrapper class. All of the Integration Object output properties can be retrieved from the output properties class.

7. At this point, you are ready to create a Web service using the tools provided in the IBM Rational SDP. Familiarize yourself with the procedure and the options available to you by reviewing the Bottom-up Web services development chapter. At a minimum, you should start the server that you are going to deploy the Web service to before continuing to create the Web service. These instructions assume that you are going to use separate steps to create your Web service, test your Web service, and create your Web service client.

8. Expand the webserviceclasses package. Right-click the wrapper class that you created, and select **Web Services > Create Web service**. The Web service type defaults to **Bottom up Java bean Web service**. Make sure that you keep this default. You can click **Finish** here or select other options that best meet your requirements, as described in the Rational SDP documentation. If you select the option, **Generate WSDL file into the project**, Rational SDP creates a Web Services Description Language (WSDL) file that describes the interfaces to your Web service. You can use this WSDL file to test your Web service using the Web

services explorer in the HATS project view. If you do not select this option, you can use a dynamically generated WSDL file to test your Web service from Services view of the Java EE perspective. When you have made all your choices, click **Finish**.

9. Go to "Testing your Web service with Web Services Explorer" for information about testing your Web service before you create a client.

Creating a Web service from EJB Access Beans

If you have created a HATS EJB project, as described in Chapter 8, "Creating and using a HATS EJB application," on page 81, you can create a Web service to use the services of the Integration Objects in the HATS EJB project. To create a Web service, you must have another HATS project, within the same enterprise archive (EAR file) as your HATS EJB project. This project is referred to as your *target project*, in which a Web service will be created from the EJB Access Beans you select. Follow these steps to create the Web service:

1. In the HATS EJB Project view, right-click on one of your Integration Objects and select **Export EJB Access Bean to HATS Project**. In the Export EJB Access Bean window, select the Integration Objects for which you want to export EJB Access Beans. Select the target project from the drop-down list of available projects. Click **Finish**.
2. In the HATS Projects view, expand the target project, the **Source** directory, and the **IntegrationObject** directory. You will see items in this directory with the name *io_name*_Access, where *io_name* is the name of the Integration Object from which the access bean was created.
3. Right click on any of these access beans and select **Create HATS Web Services Support Files**.
4. Follow the steps beginning with step 3 on page 66 in the list under "Creating a Bottom-up Web service from Integration Objects" on page 66.

Testing your Web service with Web Services Explorer

You can test your Web service in Rational SDP before you create a client application.

Before you start this process, ensure that the HATS runtime is started. You can do this by starting your server, and then, in the Servers view, choose the EAR file in which your Web service is contained, right-click, and choose **Restart**.

If you selected the option to **Generate WSDL file into the project**, see step 8 on page 67 in the list under "Creating a Bottom-up Web service from Integration Objects" on page 66, the WSDL file that you created is located within your HATS project:

- If you are creating a Web service for an IBM WebSphere JAX-RPC or JAX-WS runtime, it is located in the **Web Content/WEB-INF/Web Service Definitions** folder.
- If you created a Web service for an Apache Axis runtime, it is located in the **Web Content/Web Service Definitions** folder.

This folder contains a file called *wrapper*.wsdl, where *wrapper* is the name you gave to your wrapper class. Right-click this file and select **Web Services > Test with Web Services Explorer**.

In the right pane of the Web Services Explorer, you will see the *io_name*ProcessWS() methods for each Integration Object or EJB Access Bean included in the Web service. Click any method name to test that Integration Object. You will see a list of

the input properties that you can specify for the Integration Object. This list is based on the Integration Object's *io_name*_Input_Properties class. The only properties that must be set are the prompts that you defined when you recorded your macro. The other properties can be left blank. Refer to the Rational SDP documentation for information about using the WSDL view of Web Services Explorer to explore and test your Web service definition.

If you did not select the option to **Generate WSDL file into the project**, see step 8 on page 67 in the list under "Creating a Bottom-up Web service from Integration Objects" on page 66, you can use a dynamically generated WSDL file to test your Web service from Services view of the Java EE perspective.

Creating a Web service client

Next you can create a client application to use the Web service. Do the following:

1. Expand your project, then expand the folder appropriate for the Web service runtime you selected when you created the Web service. The WSDL file that you created is located within your HATS project:
   - If you created a Web service for an IBM WebSphere JAX-RPC or JAX-WS runtime, it is located in the **Web Content/WEB-INF/ Web Service** Definitions folder.
   - If you created a Web service for an Apache Axis runtime, it is located in the **Web Content/Web Service Definitions** folder.

   This folder contains a file called *wrapper*.wsdl, where *wrapper* is the name you gave to your wrapper class.
2. Right-click this file and select **Web Services > Generate Client** to start the Web Service Client Proxy wizard. On the first page of the wizard, you can choose all defaults except for the client project. You should choose a project other than the project that contains the Web service.

   Note: The specified **Client project** and the **Ear project** need to have the same server target as the chosen **Server**.

   You can click **Finish** at this point, or continue through the wizard. Refer to the Rational SDP documentation for information about the options in this wizard.
3. If you chose to **Test the Web service**, the wizard creates sample JSP pages, and runs the sample on the server. The output of the sample JSP pages will be displayed in the web browser. To run the sample, in the **Methods** frame, click the *io_name***ProcessWS** link to test a particular Integration Object or EJB Access Bean. In the **Inputs** frame, set the required inputs, as described in "Testing your Web service with Web Services Explorer" on page 68. The **Result** frame will display the outputs.

   The sample pages are created in your client project. In the Navigator view, expand your client project, and expand **Web Content** and **sample***wrapper***Proxy**. In the **sample***wrapper***Proxy** folder are the sample JSP pages. You can run the sample by running the `TestClient.jsp` file on the server. Double click `Result.jsp` to open it in the JSP editor. You can examine the code and copy code from the sample into a HATS business logic class, which can be run from a screen customization to invoke your Web service and use the output in a transformation or in some other way.

Creating a Top-down Web service that includes Integration Objects

Your development task may involve creating a top-down Web service, where you have been given a WSDL that has the specified signature of the Web service. Your task is to create the Web service implementation, part of which involves interacting with an existing terminal application.

If the input parameters in the WSDL either contain the required input properties for an Integration Object, or based on the input parameters, you can write code that will set the required Integration Object input properties, you can use Integration Objects directly in your Web service implementation.

The Rational SDP contains tools to create a skeleton Java bean from an existing WSDL. You can then interact with Integration Objects directly from this skeleton Java bean. Refer to Chapter 6, "Programming with Integration Objects," on page 53 to learn more. The one caveat is that you should use the processRequest () method to invoke the Integration Object in a Web services run time environment.

Refer to Chapter 1, "Introduction," on page 1

Programming with Web Services Integration Objects and EJB Access Beans

Integration Object chaining with Web Services

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

EJB Access Bean chaining with Web Services

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

Special considerations with chaining Web Services

If you use chained Integration Objects within a HATS Web service, you are creating a stateful Web service. HATS runtime does not store data between invocations of chained Integration Objects in the same Web service. However, HATS runtime does require that the next in chain Web service invocation is routed back to the same instance of the HATS runtime so that the next in chain IO will use the same Telnet connection.

If you deploy a stateful HATS Web service using JAX-RPC support to a cluster, to enable all Integration Objects in the chain to use the same Telnet connection, do one of the following:

- Create the HATS Web services from EJB Access Beans. The HATS EJB is a stateful session EJB, so the EJB client always interacts with the same HATS EJB instance.
- Configure the scope of the Web service to be session (this can be done in the development environment or after deployment), and use SESSION_MAINTAIN_PROPERTY in the Web service client runtime to maintain the session across invocations, therefore ensuring HTTP session affinity. For more information about stateful Web services, see the following article: http://www-128.ibm.com/developerworks/webservices/library/ws-tip-

stateful.html. Note that WebSphere APAR PK35259 may be required, as it fixes an issue with SESSION_MAINTAIN_PROPERTY.

If you deploy a stateful HATS Web service using JAX-WS support to a cluster, to enable all Integration Objects in the chain to use the same Telnet connection, do one of the following:

- Create the HATS Web services from EJB Access Beans. The HATS EJB is a stateful session EJB, so the EJB client always interacts with the same HATS EJB instance.
- Use HTTP session management support following the directions found in the WebSphere Application Server Knowledge Center at http://publib.boulder.ibm.com/infocenter/wasinfo/v8r0/index.jsp?topic=/com.ibm.websphere.express.doc/info/exp/ae/twbs_enablesecconvcluster.html.

Updating Web services

The Update wizard is only enabled for Web service classes that have been created with HATS 7.0.0.2 or later. If you are using an earlier version, then you must update manually.

To update the HATS Web service support classes, do the following:

1. Select an existing Web service wrapper class to update.
2. The Update Web service wizard is invoked with the Web service class preselected.
3. If you do not need to change the selection of Integration Objects or EJB Access Beans included in the Web service, you can select **Finish** to regenerate the Web service support classes, otherwise select **Next** to continue to the next page of the wizard.
4. Modify the list of resources (Integration Objects and EJB access beans) included in the Web service wrapper class . If you want to change the input and output properties that are exposed by the Web service for a given resource, select the resource and click **Properties**.
5. On the Choose Properties page, select the input and output properties that you want exposed in the Web service. If you wish, you can provide an alias name for the property in the Alias Name field. Use the **Select All**, **Deselect All**, and **Select Default** buttons to help in selecting the properties. Click **OK**.

   Notes:

   a. Defaults for single Integration Objects and EJB access beans are all macro prompts for input properties and all macro extracts for output properties.
   b. Defaults for chained Integration Objects are all macro prompts plus hPubLinkKey (which is required) for input properties and all macro extracts plus hPubLinkKey (which is required) for output properties.
   c. Defaults for chained EJB access beans are all macro prompts plus hPubLinkKey and hPubAccessHandle (both of which are required) for input properties and all macro extracts plus hPubLinkKey and hPubAccessHandle (both of which are required) for output properties.
   d. To specify that Integration Object connection overrides be exposed to the Web service, you must select the appropriate properties, for example the hPubConnectionOverrides or hPubStartPoolName properties, which are not selected by default. For more information, see “Specifying Connection Overrides” on page 57.
   e. For more information about the use of other Integration Object methods, see Chapter 6, “Programming with Integration Objects,” on page 53.

f. For considerations when using bidirectional language support, see Support of bottom-up Web services in the *HATS User's and Administrator's Guide*.

6. Click **Finish**.

The HATS Web service support files are regenerated and compiled. Note that you still must use the Rational SDP wizards to update your Web services once you have updated the HATS Web services support files.

Web services for JAX-WS runtime considerations and limitations

If you plan to create HATS Web services for the JAX-WS runtime, be aware of the following:

- When you generate a test client, the web_service_nameService.java file is created with a hardcoded, absolute location of the WSDL file. If you deploy the test client to a different machine, update the file to contain the correct WSDL file location.
- The WSDL file that you elect to generate for your Web service contains a URL location of the Web service. This URL points to the local host and port. Update the local host and port in the URL when you deploy the Web Service to another server.

Creating RESTful Web services

As with traditional Web services, RESTful Web services are created from HATS Integration Objects and their input and output properties. With traditional Web services, you define the location and input and output properties of the Web service resources in a WSDL file. However, with RESTful Web services, you define access to the Web service resources using URIs to represent the resources, HTTP methods to operate on the resources, and HTTP header information to define the message formats.

In the following figure is an architectural view of HATS support for RESTful Web services.

1. The HATS RestServlet receives from a client an HTTP request with URI and an HTTP method.
2. The RestServlet routes the request by mapping information in the URI and the HTTP method to a JAX-RS resource for an Integration Object.
3. The JAX-RS resource receives the request, reads the parameters in the request, initializes the Integration Object, sets prompts (input properties), and runs the Integration Object.
4. The JAX-RS resource receives extracts (output properties) from the Integration Object, generates a response, and returns it to the RestServlet.
5. The RestServlet responds to the client.

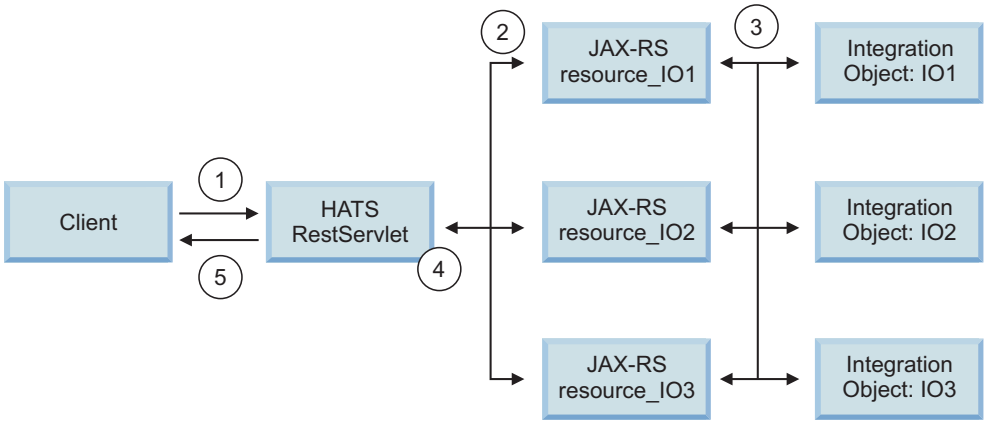


Figure 20. HATS RESTful Web service architecture

Note: To run a HATS Integration Object, the client must call one Integration Object at a time. For chained Integration Objects, the client must call one Integration Object, then call the next, passing the link key, or you must modify the wrapper to call one Integration Object after another, passing the link key.

HATS provides tools you can use to create JAX-RS resources for your Integration Objects, and mappings so that URIs and HTTP methods in HTTP requests can be mapped to the correct JAX-RS resources.

Creating RESTful service JAX-RS resources

To create a JAX-RS resource for an Integration Object, in the HATS Projects view:

1. Expand the project that contains the Integration Object you wish to use, then expand the **Source** folder and the **IntegrationObject** folder.
2. Right-click the Integration Object and select **Create RESTful Service Files**. The Create RESTful Service Files wizard opens.
3. On the Specify JAX-RS Resource class name and Integration Object page, in the Name field, specify the name of the JAX-RS resource class to generate in the Source folder. The name must follow generic Java class name syntax.
4. Optionally select whether to **Overwrite resources without warnings**.
5. Click **Next**.
6. On the Configure JAX-RS Resource class page, in the URI Suffix field, specify a suffix to create the complete URI to use in mapping to your JAX-RS resource. For example, if the host where the HATS RESTful service is installed is www.myHost.com, the HATS project (application) name is myApp, and the URI Suffix is mySuffix, then the resulting URI for the JAX-RS resource is http://www.myHost.com:9080/myApp/rest/mySuffix.

   Note: The combination of URI plus HTTP method must be unique among all of the JAX-RS resources that you define. If not, the JAX-RS runtime picks only one of the resource functions to invoke based on a priority algorithm using the combination of consumes and produces content types (see "Customizing RESTful service JAX-RS resource methods" on page 75).
7. Click the **Add** button to add at least one method to the resource class. The Define JAX-RS RESTful Service Method wizard opens.

8. In the HTTP Method field, from the drop-down list, select the HTTP method (GET, POST, PUT, DELETE) you want to use in combination with the URI to map to this JAX-RS resource method.
9. Select the **Use Integration Object** box if you want this method to invoke an Integration Object. Clear the box if you want to generate an empty method without the Integration Object handling code. Do this to create your own customized method.
10. If you select the **Use Integration Object** box, from the drop-down list, select the Integration Object you want this method to invoke.
11. Click **Next**.
12. If you selected the **Use Integration Object** box, the Choose Integration Object properties page displays. Select the input and output properties that you want exposed as parameters for the RESTful Web service. If you wish, you can provide an alias name for the property in the Alias Name field. Use the **Select All**, **Deselect All**, and **Select Default** buttons to help in selecting the properties.

    Notes:

    a. Defaults for single Integration Objects and EJB access beans are all macro prompts for input properties and all macro extracts for output properties.
    b. Defaults for chained Integration Objects are all macro prompts plus hPubLinkKey (which is required) for input properties and all macro extracts plus hPubLinkKey (which is required) for output properties.
    c. Defaults for chained EJB access beans are all macro prompts plus hPubLinkKey and hPubAccessHandle (both of which are required) for input properties and all macro extracts plus hPubLinkKey and hPubAccessHandle (both of which are required) for output properties.
    d. To specify that Integration Object connection overrides be exposed to the Web service, you must select the appropriate properties, for example the hPubConnectionOverrides or hPubStartPoolName properties, which are not selected by default. For more information, see "Specifying Connection Overrides" on page 57.
    e. For more information about the use of other Integration Object methods, see Chapter 6, "Programming with Integration Objects," on page 53.
13. Click **Next**.
14. On the Configure JAX-RS Resource Method page, defaults are set that reflect the HTTP method and Integration Object you selected. For more information about these method settings and how to customize them, see "Customizing RESTful service JAX-RS resource methods" on page 75. For considerations when using bidirectional sessions, see Support of RESTful Web services in the *HATS User's and Administrator's Guide*.
15. Click **Finish**.
16. On the Configure JAX-RS Resource class page, in the Methods section, are listed the methods you have added. Use the **Add** button to add another method, use the **Edit** button to edit the selected method, and use the **Remove** button to remove the selected method.
17. After you click **Finish**, HATS creates the JAX-RS resource file and stores it in the **Source\restfulserviceclasses** folder in your project.
18. In addition, HATS adds the JAX-RS resource to the list of services for the RestServlet servlet to scan for a match. This list is maintained in the WEB-INF/wink-resources.lst file viewable in the Navigator view.

Notes:

a. HATS creates the WEB-INF/wink-resources.lst file when the first JAX-RS resource is created in a project.

b. HATS updates the default WAR class loader policy of your HATS application to **Single class loader for application**.

c. HATS also adds the JAX-RS facet to the project.

d. If you create a JAX-RS resource file manually, not using the Create RESTful Service Files wizard, you must update the WEB-INF/wink-resources.lst file to map the JAX-RS resource to the RestServlet servlet.

Updating RESTful service JAX-RS resources

To update a JAX-RS resource for an Integration Object, in the HATS Projects view:

1. Expand the **Source** folder and the **restfulserviceclasses** folder.
2. Right-click the JAX-RS resource and select **Update RESTful Service Files**. The Update RESTful Service Files wizard opens.
3. You can make changes for the JAX-RS resource by following the instructions described in "Creating RESTful service JAX-RS resources" on page 73.

Customizing RESTful service JAX-RS resource methods

If you want to customize a HATS RESTful service JAX-RS resource method, for example, to handle content types other than application/xml and application/json, or to define method parameters, follow these steps:

1. Follow the instructions for "Creating RESTful service JAX-RS resources" on page 73 or "Updating RESTful service JAX-RS resources."
2. On the Configure JAX-RS Resource class page, click **Add** to add a new method or click **Edit** to edit the highlighted method.
3. Set the HTTP method, Integration Object, and Integration Object properties as appropriate.
4. On the Configure JAX-RS Resource Method page, in the Method Name field, specify the Java method name.
5. In the URI Suffix field, add any additions to the path, and add any definitions for URI parameters as PathParam parameters in the format {name}.
6. In the Return Type field, add or change the Java type returned by this method. The default is `restfulserviceclasses.`*`JAX-RS resource class name`*`+ _`*`resource method name`*`+ _Output_Properties`.
7. In the Consumes field are listed the content types (Internet media types, or MIME types) supported for input in the HTTP request body. Add or change the content types of request messages supported. Multiple content types can be specified separated with commas. HATS supports application/xml and application/json as defaults. This is only applicable for HTTP Methods POST or PUT. The HTTP GET and DELETE methods do not have HTTP request bodies and do not support the Consumes field. For these methods, input must instead be supplied as URI parameters with content type application/x-www-form-urlencoded. For more information, see "Handling content" on page 77.
8. In the Produces field are listed the content types supported for output in the HTTP response body. Add or change the content types of response messages supported. Multiple content types can be specified separated with commas. HATS supports application/xml and application/json as defaults. For more information, see "Handling content" on page 77.

9. In the Method Parameters fields add or change the input parameters supported by this method. Use the **Add**, **Edit**, and **Remove** buttons to create, modify, and delete the input parameters for this method.
10. If you create or modify an input parameter, on the Method Parameter page complete the following fields. For more information see the JAX-RS specifications at http://jcp.org/aboutJava/communityprocess/final/jsr311/index.html.

    Parameter Type
    Select from the drop-down one of the following types as defined in the JAX-RS specification:
    - **Entity** - a non-annotated parameter, for example, a normal Java object, extracted from the request entity body.
    - **CookieParam** - parameter value is to be extracted from an HTTP cookie.
    - **FormParam** - parameter value is to be extracted from an HTML form parameter.
    - **HeaderParam** - parameter value is to be extracted from an HTTP header.
    - **MatrixParam** - parameter value is to be extracted from a URI matrix parameter.
    - **PathParam** - parameter value is to be extracted from the request URI path.
    - **QueryParam** - parameter value is to be extracted from a URI query parameter.
    - **Context** - parameter value is to be extracted from a Web application context.

    Parameter Name
    Name of the parameter. Used for CookieParam, FormParam, HeaderParam, MatrixParam, PathParam, and QueryParam.

    Default Value
    Default value if the parameter cannot be found. Used for CookieParam, FormParam, HeaderParam, MatrixParam, PathParam, and QueryParam.

    Type Input parameter Java type

    Name Input parameter name

Note: Each JAX-RS resource must contain at least one method. If the **Use Integration Object** option is selected, the method contains the HATS RESTful wrapper that maps the input and output parameters for the Integration Object and invokes the Integration Object. By default, HATS maps the parameters to the Integration Object as follows:
- For HTTP PUT or POST, and the input parameter is mapped as a single entity (Entity).
- For HTTP GET or DELETE, and the input parameter(s) are mapped as query parameters (QueryParam).
- The output return type is always an entity (Java object).

If you modify the method parameters as described in this section, HATS does not map the parameters for the Integration Object and does not perform other handling for the modified Java method. In this case, you must edit the JAX-RS resource class file and implement this yourself.

Handling content

When you create a JAX-RS resource method for an Integration Object, the Consumes field specifies the content types (Internet media types, or MIME types) the JAX-RS resource method supports as input in a HTTP request body. The Produces field lists the content types the JAX-RS resource method supports as output in a HTTP response body.

The content type of an HTTP request or response is defined in the Content-Type HTTP header field. An HTTP request can also use the Accept HTTP request header field to specify the content types it can accept in the response.

For more information, see the references below:

- For HTTP specifications, see http://www.w3.org/Protocols/.
- For JAX-RS specifications, see http://jcp.org/aboutJava/communityprocess/final/jsr311/index.html.
- For JSON format definitions, see http://www.json.org/.

Content type examples

The following examples summarize handling content in HTTP requests and responses. These examples are intended to show concepts and not exact HTTP protocols.

HTTP GET and DELETE requests specify all of their input parameters on the URI using standard encoding. There is no body in such requests. The content type of the requests for these is always x-www-form-urlencoded, since that is all that is allowed for GET and DELETE requests by the HTTP protocol. The client indicates what content type it can accept in the response by supplying an Accept header.

In the GET request below, the parameter is passed as a query string on the URI. The content type acceptable in the response is specified in the Accept header field as application/xml.

```
GET http://www.myHost.com:9080/myApp/rest/myCustomer?name=john%20doe
Accept: application/xml
```

All HTTP responses carry their data in their body, no matter what kind of request prompted the response. Responses begin with header fields. The Content-Type header field in a response tells the client what type of data is contained in the body below. HTTP allows many different content types in responses, but HATS JAX-RS supports only application/xml and application/json. After the header fields and a blank line, the body begins, and contains the data in the specified format.

In the GET response below, the Content-Type header field specifies the type of data contained in the response body.

```
Content-Type: application/xml

<customer>
 <firstname>John</firstname>
 <lastname>Doe</lastname>
 <accountnumber>111111</accountnumber>
</customer>
```

By convention, a POST request typically asks to create something. The data required to fulfill the request is carried in the request body. The Content-Type indicates to the server the format of the request data. The Accept header again specifies the desired response format.

In the POST request below, the content type is specified in the Content-Type header field as application/xml and the content is supplied in the request body. The Accept header field specifies the acceptable format for data in the response, as application/json.

```
POST http://www.myHost.com:9080/myApp/rest/myCustomer
Content-Type: application/xml
Accept: application/json

<customer>
 <firstname>Jane</firstname>
 <lastname>Doe</lastname>
 <accountnumber>222222</accountnumber>
</customer>
```

In the POST response below, notice the format of the response does not have to match the format used in the request.

```
Content-Type: application/json

{"message":{
  "type": "resultCode",
  "value": "Jane Doe account created successfully"
}}
```

The HATS JAX-RS output format rule is to respond using the format specified in the Accept HTTP request header unless the client specifies a format using the URI query parameter `alt` as shown below:

```
POST http://www.myHost.com:9080/myApp/rest/myCustomer?alt=application/xml
```

Customizing the response header

Sometimes you might need to customize the response header of your RESTful service. For example, to handle response headers that use Shift_JIS encoding, you must modify the REST resource class file to change the return type to javax.ws.rs.core.Response and change the content type or custom header. For example, if you want to use Shift_JIS as the default charset in a POST method with a FormParam, perform the following steps:

1. Specify WebSphere Application Server encoding as Shift_JIS. For information about how to do this, see http://publib.boulder.ibm.com/infocenter/wasinfo/v8r0/index.jsp?topic=/com.ibm.websphere.base.doc/info/aes/ae/trun_svr_utf.html.
2. Change the return type of the method in the REST resource class to javax.ws.rs.core.Response. For example, change the following code from:

```
@POST
@Consumes({"application/x-www-form-urlencoded"})
@Produces({"application/xml"})
public xxx_Output_Properties invoke(@FormParam("id") String id, @FormParam("yyy") String yyy)
{
                    :
                    :
  return outputToClient;
}
```

   To:

```
@POST
@Consumes({"application/x-www-form-urlencoded"})
@Produces({"application/xml"})
public javax.ws.rs.core.Response invoke(@FormParam("id") String id, @FormParam("yyy") String yyy)
{
                    :
                    :
  return javax.ws.rs.core.Response.ok(outputToClient, "application/xml;charset=Shift_JIS").build();
}
```

HTTP status codes

The following HTTP status codes are returned from HATS RESTful Web services:

200 Successful. No error.

400 General error. For example, incorrect content in the request body.

404 URI not found.

405 Method not allowed. For example, the service supports HTTP POST and PUT, but the URI requests HTTP DELETE.

406 Unsupported response format is requested. Supported response format is returned in the Accept header.

415 Unsupported request format. The format used for the request in invalid. Used for HTTP POST and PUT when an unsupported value is supplied in the request's Content-Type HTTP header field.

500 Internal server error, for example an Integration Object execution error. Throws WebApplicationException.

For more information about HTTP status codes, see http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html.

JAX-RS RESTful services considerations and limitations

Following is a list of considerations and limitations when using HATS support for JAX-RS RESTful Web services:

- HATS runtime support for RESTful Web services requires the JAX-RS support included in the Feature Pack for Web 2.0 for WebSphere Application Server Version 7.0. WebSphere Application Server V8.0 and V8.5 include the required JAX-RS support and no additional feature packs are necessary. For more details, see "System Requirements for Host Access Transformation Services" at http://www.ibm.com/support/docview.wss?uid=swg27011794.
- HATS JAX-RS RESTful services are not compatible with Portal.
- To create customized providers, you must follow the JAX-RS specification, see http://jcp.org/aboutJava/communityprocess/final/jsr311/index.html, and refer to the Apache Wink implementation, see http://incubator.apache.org/wink/.

Chapter 8. Creating and using a HATS EJB application

As an alternative to the usual HATS project, which transforms one or more host applications and presents Web pages to an end user, you can build a HATS Enterprise JavaBeans (EJB) project. A HATS EJB project provides access to the host interactions, which are encapsulated in Integration Objects. The Integration Objects are needed to provide host data to an EJB client, which can be another HATS application, a user-written Java program, a Web application, or even another EJB. A HATS EJB project enables you to separate the collection of the host data from its presentation. If you have one or more Integration Objects that you want to be able to invoke from several HATS projects, you can include them in a HATS EJB project and call them from several other HATS projects. When you have finished configuring your HATS EJB project, you can assemble it as a HATS EJB application.

Notes:

1. HATS EJB project support is deprecated in HATS V9.5. While support for HATS EJB projects continues for now, IBM reserves the right to remove this capability in a subsequent release of the product. Alternatives are:
   - Use Web services to access your Integration Objects. For more information see Chapter 7, "Developing Web services," on page 65.
   - Create custom EJB beans to access your Integration Objects. For more information see "Using an Integration Object in an EJB container (from your own EJB)" on page 99.
2. This section assumes that you are familiar with basic EJB concepts. If not, refer to the Rational SDP documentation on this subject, or the WebSphere Knowledge Center for your version and edition of WebSphere Application Server.
3. You cannot include an Integration Object that is configured to use Web Express Logon in a HATS EJB project.
4. If you have developed EJB projects in Host Publisher, you can import them into HATS projects. Refer to *HATS User's and Administrator's Guide* for information about importing Host Publisher projects into HATS.

A HATS EJB application does not interact directly with an end user, nor does it transform host screens. It does not contain the resources that are included in a HATS project: screen customizations, templates, or transformations. A HATS EJB application contains one or more Integration Objects, whose services it makes available to calling programs.

When you create HATS EJB projects, a new view is opened in HATS Toolkit. This view is called the HATS EJB Project view. It is analogous to the HATS Projects view, but it contains only HATS EJB projects. It shows the EJB resources that are described in this section. To work with your HATS EJB project, you can use the HATS EJB Project view in the same ways that you use the HATS Projects view for other HATS projects. For example, you can click the project name in the HATS EJB Project view and then click the **host terminal** icon to open the host terminal for the project's main connection, in order to record macros.

A HATS EJB application is an enterprise archive (EAR file) made up of one or more Integration Objects, whose services it makes available to an EJB client and typically consists of these parts:

- A HATS project (Web module), which includes:
  - One or more EJB Access Beans. The EJB Access Bean locates the HATS EJB, creates an instance of the EJB, and invokes the main business method of the EJB, passing as input an Integration Object properties object, which contains the Integration Object properties that are set by the JSP.
  - One or more JSPs that interface with an EJB Access Bean as if driving an Integration Object locally.

  These resources appear in the HATS Projects view.
- A HATS EJB Project (EJB Module) that consists of:
  - The HATS EJB. The HATS EJB is implemented as a stateful session EJB.
  - One or more Helper objects. The Helper object is invoked with the input parameters, and causes the actual Integration Object function to be driven. When the Integration Object completes, the new instance of an Integration Object properties object is returned to the EJB Access Bean.
  - One or more Integration Objects (.java files)
  - One or more connection definitions (.hco files)
  - Optionally, one or more connect and disconnect macros (.hma files)
  - One or more macros from which Integration Objects can be created (.hma files)
  - An application file (application.hap)
  - EJB support files (*.java files)

  These resources appear in the HATS EJB Project View.

In addition to providing the HATS EJB, HATS EJB support generates an EJB Access Bean for each Integration Object. This EJB Access Bean has the same signature as the Integration Object, allowing applications and JSPs to be developed to drive an Integration Object using the EJB Access Bean. Because the EJB Access Bean has the same signature as the Integration Object, the EJB Access Bean can be used in client-side code exactly as the original Integration Object would be used. Therefore, the client can be:

- A JSP or a servlet that uses one or more EJB Access Beans in a Web application where the Integration Objects execute in an EJB container
- A custom EJB that uses one or more EJB Access Beans to execute the Integration Objects in an EJB container
- A Java application running in the WebSphere Application Client execution environment

Figure 21 shows the links between a HATS EJB application and other applications that use its services.

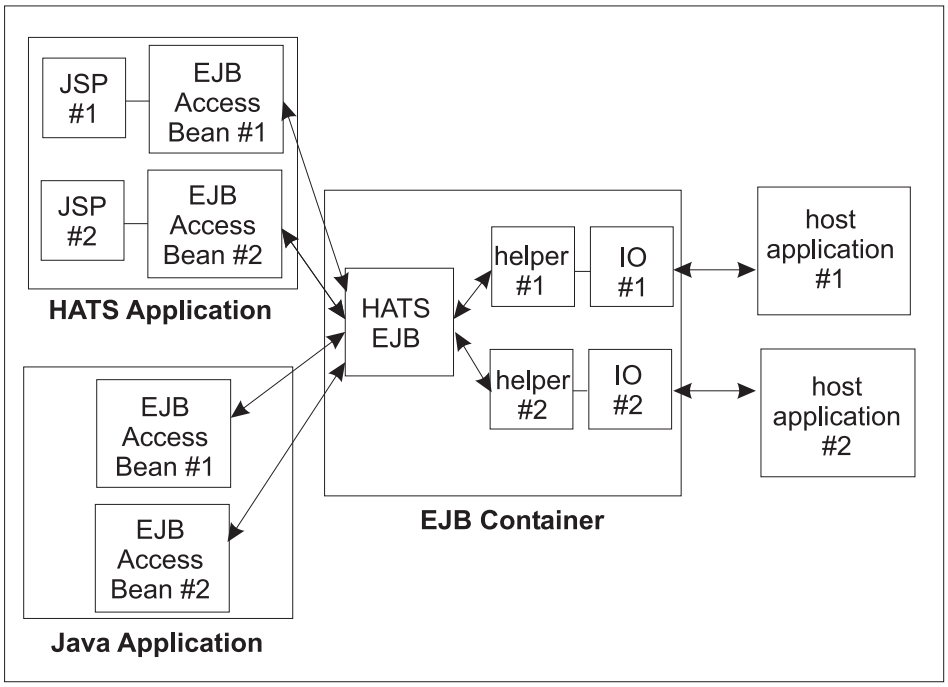


Figure 21. A HATS EJB project

Creating a HATS EJB project

Follow these steps to create a HATS EJB project:

1. Start with a HATS project, which will ultimately contain the EJB Access Beans. This is referred to as the *target* HATS project.
2. Click **File > New > Project > HATS > HATS EJB Project**. Be sure to put the HATS EJB project in the same EAR as your HATS target project. Define one host connection in the wizard.
3. If you want to use Integration Objects that were created with Host Publisher Studio, import them into your project. Click **File > Import**, select **Host Publisher IO**, and complete the wizard. You do not need to define connections or record macros for these Integration Objects.
4. Define the connections to any additional host machines from which you will extract data. Refer to *HATS User's and Administrator's Guide* for information about defining connections.
5. If any of your host connections require connect and disconnect macros, use the host terminal to record those macros. Then use the connection editor to update each affected connection to specify the names of the connect and disconnect macros that are used on that connection. Refer to *HATS User's and Administrator's Guide* for information about recording connect and disconnect macros.
6. If you have not already created Integration Objects, connect to the host applications from the HATS EJB Project view and follow the instructions in *HATS User's and Administrator's Guide* to create Integration Objects.
7. When your HATS EJB project contains all the connections, macros, and Integration Objects you require, you are ready to export EJB Access Beans to your target HATS project. To do this, right click any Integration Object in your EJB project in the HATS EJB Project view and select **Export EJB Access Bean to**

HATS Project. You can then select the target HATS project, and the Integration Objects for which you want EJB Access Beans created.

8. Now you are ready to work with the EJB Access Beans that you have just created in the target HATS project. The EJB Access Beans are in the **source** folder of the target HATS project, in the IntegrationObject package, in files with a name pattern of *io-name*_Access.java. You can create HATS business logic to drive an EJB Access Bean; the process is the same as driving an Integration Object from business logic, which is described in Chapter 2, "Adding business logic," on page 3. You can use Rational SDP capabilities to Insert Beans and Insert Results from Beans onto JSP pages. You cannot generate HATS Model 1 or Struts Web pages directly from an EJB Access Bean.
9. Assemble the projects into an enterprise archive (EAR file), as described in HATS Getting Started. Deploy it to WebSphere Application Server.

You can also use the EJB Access Beans in Java code that you write yourself. See "Programming with EJB Access Beans" on page 85 for information about using the EJB Access Beans in code that you write.

The EJB Access Beans are created based on the macro that was recorded and made into an Integration Object. If you modify the macro in such a way that it changes the signature of the Integration Object, and redeploy the HATS EJB project, you must re-export the matching EJB Access Beans and modify the code or JSPs that use them.

Storing a HATS EJB project in a repository

If you store a HATS EJB project in a repository, such as CVS, some directories are not stored. You must store the contents of these directories manually. After storing your project, you must also store all the files in these directories. These files will not be included if a HATS EJB project is stored as an archive file.

- ejbModule\com\ibm\ejs\container
- ejbModule\com\ibm\HostPublisher\EJB
- ejbModule\com\ibm\websphere\csi
- ejbModule\org\omg\stub\javax\ejb

Creating EJB Access Beans automatically

To make it easier to use EJB Access Beans in a project that is not a HATS project, refer to the preference **Automatically generate EJB Access Bean in HATS EJB project when an Integration Object is created** on the HATS Preferences page.

If you click this preference, an EJB Access Bean is created in your HATS EJB project whenever you create an Integration Object. If you have selected the **Automatically generate Integration Object when saving macro** preference, an EJB Access Bean is created along with the Integration Object when you create or modify a macro.

You can then copy the EJB Access Bean directly to your other project without having to export it to a HATS project. You must still follow the steps listed in "Using EJB Access Beans with Java application clients" on page 86 to add the EJB reference and EJB environment variable in your other project.

EJB Access Beans are not created automatically when you migrate a project.

For more information about using HATS preferences, refer to *HATS User's and Administrator's Guide*.

Programming with EJB Access Beans

When you create an Integration Object in a HATS EJB Project in HATS Toolkit, EJB support files are generated. One of the files generated is the EJB Access Bean. The EJB Access Bean is a Java bean that has the same signature as the Integration Object that the Access Bean supports. Your application will use the EJB Access Bean to interact with the Integration Objects that are included in the HATS EJB application.

Using EJB Access Bean methods

HATS Integration Objects contain Java methods that you can use when programming with Integration Objects. These methods are also available when programming with EJB Access Beans. In addition to the methods described in "Integration Object methods" on page 54, you can use the following methods when programming with EJB Access Beans:

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

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

EJB Access Bean chaining

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

hPubAccessHandle
: Contains the handle for the HATS EJB instance corresponding to the Integration Object chain

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

EJB Access Bean chaining in a Web container: When EJB Access Beans are used in a Web module with the `doHPTransaction` method, the EJB Access Bean saves the values of the **hPubAccessHandle** and **hPubLinkKey** properties in the HttpSessionObject that is associated with the client session. The properties are retrieved by subsequent EJB Access Beans during processing of the Integration Object chain.

The EJB Access Bean for the first Integration Object in a chain creates an instance of the HATS EJB. That instance is used for all EJB Access Beans for the subsequent Integration Objects in the chain. After the processing of the EJB Access Bean for the last Integration Object in a chain, the HATS EJB instance corresponding to that Integration Object chain is removed. Thus, when using the `doHPTransaction` method in a Web module with EJB Access Beans, chaining is managed within the EJB Access Bean.

EJB Access Bean chaining outside a Web container: When EJB Access Beans are processed outside of a Web container (for example, directly from a Java program), the calling program must retrieve the values of the hPubAccessHandle and hPubLinkKey properties. The properties can be retrieved after the EJB Access Bean

for the first Integration Object in the chain is processed. The calling program must set the two properties for all subsequent EJB Access Beans in the Integration Object chain before they are processed.

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

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

   ```
   EJBChain1.set<i>Xyz</i>(String)
   ```

   where *Xyz* is the name of your input variable.
3. Invoke the EJB Access Bean to perform its task, for example running a macro, using the method:

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

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

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

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

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

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

    ```
    EJBChain2.set<i>Xyz</i>(String)
    ```

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

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

    ```
    EJBChain2.getHPubErrorOccurred()
    ```

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

Using EJB Access Beans with Java application clients

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

- Java EE application client, which supplies a container that provides access to system services for the application client code.
- Java thin application client, a lightweight, downloadable Java application runtime that is capable of interacting with enterprise beans.

For more information about these client modules, refer to the WebSphere Knowledge Center documentation.

The following sections assume that you have chosen to use a Java EE application client. If you are using a Java thin application client, consult the WebSphere Knowledge Center for the instructions that are appropriate for your WebSphere release.

Complete the following steps to use EJB Access Beans with a Java application client:

1. Start with a HATS EJB project that contains one or more macros, Integration Objects, and other resources, and a HATS project to which you have exported EJB Access Beans from the HATS EJB project.
2. Create a Java EE application client as documented in the Knowledge Center for your version and edition of WebSphere Application Server. To follow these instructions, choose to generate a Main class, and a deployment descriptor. Put this new project into the same enterprise archive (EAR file) as the HATS EJB project. Add these jar files as dependent jars:
   - The EJB .jar file of the HATS EJB project
   - hatscommon.jar
3. Move your EJB Access Beans to the application client project. Copy the IntegrationObject folder that holds the EJB Access Beans from the HATS project to the **appClientModule** directory of the application client project.
4. Build your test client in the Main class that was generated in your application client by implementing the code to instantiate your EJB Access bean and run it.
5. If you are using the Java EE application client, add the EJB reference and environment variables to the client deployment descriptor:
   a. Open META-INF\application-client.xml.
   b. On the **Design** tab of the Application Client Descriptor Editor, do the following:
      1) Click **Add** and choose to add an environment variable:
         - **Name**: Copy the value of the last part of the HPUBEJB2_REFERENCE string in the EJB Access Bean into this field. For example, it might say `HPUBEJB2_REFERENCE1318657356`.
         - **Type**: java.lang.String
         - **Value**: ejb/*ejb_project_name*, where *ejb_project_name* is the name of your HATS EJB project.
      2) Click **Add** and choose to add an EJB reference:
         - **EJB Reference name**: ejb *ejb_project_name*, where *ejb_project_name* is the name of your HATS EJB project.
         - **EJB Reference type**: Session
         - **Home**: com.ibm.HostPublisher.EJB.HPubEJB2Home
         - **Remote**: com.ibm.HostPublisher.EJB.HPubEJB2
         - **EJB link**: *ejb_project_name*.jar# *ejb_project_name*, where ejb_project_name is the name of your HATS EJB project.
6. Publish the EAR file to your test server.
7. Test the application client:
   a. Open the Debug perspective.
   b. Start the application server.

c. Click the **Debug** drop-down list, and select **Debug** to define and start the test environment for the Java EE application client that you created:
   1) Click **WebSphere Application client**.
   2) Click **New**.
   3) Click the **Application** tab and complete the fields:
      - In the **Name** field, type a name for your test configuration that is unique to your application client.
      - Select the WebSphere server type.
      - Select the EAR that your application client resides in.
      - Select your application client module.
      - Enable the application client to connect to a server and choose either a specific server or a provider URL.
      - Click **Apply**.
      - Click **Debug** to start the test environment. You will see the output of your test client in the console.

8. Export the application client. From the Navigator view, right click the name of the application client project and select **Export**. Export it as an EAR file.

Running your application client

After you have completed the steps in the previous section, follow these steps to run your application client:

1. Copy your EAR file into the bin directory of the application client.
2. Open a command prompt and enter `Run launchClient` *`ejb_project_name`*`EAR`, where *ejb_project_name* is the name of the ear file that you copied into the bin directory of the application client. This will run your application client. If you do not see the output you expect, check launchClient.bat to make sure that DEFAULTSERVERNAME is set to your test server and SERVERPORTNUMBER is set to the Bootstrap port of the test server. The default Bootstrap port is 2809, but might be different depending on the WebSphere configuration. The Bootstrap address is the combination of the Bootstrap Host and Bootstrap port, with a colon (:) between them. To find the Bootstrap port of the WebSphere Application Server, open the WebSphere Administrative Console and look for **Servers > Application servers >** *servername* **> ports > BOOTSTRAP ADDRESS**.

Running EJB application clients in a clustered environment

To run an EJB application client in a clustered environment, there are two options:

- To run the application client on a specific cluster member, follow the instructions in "Running your application client," but modify DEFAULTSERVERNAME and SERVERPORTNUMBER as follows:
  - Set DEFAULTSERVERNAME to the host name of the cluster member where the EJB is to be run.
  - Set SERVERPORTNUMBER to the Bootstrap port of the cluster member where the EJB is to be run. The Bootstrap address is the combination of the Bootstrap Host and Bootstrap port, with a colon (:) between them. The Bootstrap port can be found by opening the WebSphere Administrative Console and look for **Servers > Application servers >** *cluster-member-name* **> ports > BOOTSTRAP ADDRESS**.
- To run the application client on any available cluster member within the cluster, set up the environment to specify the appropriate JNDI naming configuration and parameters. Refer to the WebSphere Application Server Knowledge Center

at http://www.ibm.com/support/knowledgecenter/SSAW57 and search for `Using application clients` for instructions on how to configure the environment.

Chapter 9. Integration Objects - advanced topics

This chapter discusses customizing Integration Object Java code, Integration Object templates, including choosing and modifying them, using Integration Objects in a WebSphere Java EE application, and a connection management API.

Customizing Integration Object Java code

When you create an Integration Object from a macro or when HATS creates an Integration Object automatically when you save a macro, HATS uses Integration Object *templates* to create the Integration Object. These templates contain the Java code that is included in each Integration Object.

HATS enables you to modify how an Integration Object interacts with the underlying subsystems, at the Java code level, to perform additional functions.

There are two types of Java coding templates: default templates and customizable templates. The templates are stored in

```
<shared_install_directory>\plugins\com.ibm.hats_nnn\predefined\
IOTemplates\
```

where *shared_install_directory* is the shared resources directory where you installed the HATS offering using IBM Installation Manager and *nnn* is the version and build level of HATS.

The default templates are HPubTemplateHODBean.Default and HPubTemplateHODBeanInfo.Default. The customizable templates are HPubTemplateHODBean.Customize and HPubTemplateHODBeanInfo.Customize. The default templates contain Java code that is independent of the HATS and Host On-Demand code. Integration Objects that are created using the default templates do not need to be recompiled and redeployed if the HATS or Host On-Demand code changes for enhancements or service.

The default templates call methods from the superclass. The customizable templates contain the methods, which you can customize. You can modify the HPubTemplateHODBean.Customize template to add function to your Integration Objects. For example, you might want to find out not just the text on the host screen, but also its characteristics, such as color or highlighting.

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

The templates that are used by HATS are different from those that are used by Host Publisher. If you modified an Integration Object template in Host Publisher, you must make the same changes to the templates that are provided by HATS and recreate your Integration Objects in order to achieve the same functions in your Integration Objects. When you import or migrate a Host Publisher Integration

Object that was created with a modified template, or an application using such an Integration Object, you will see a message indicating that the Java bean was created from a customized template.

Choosing Integration Object templates

If you do not need to modify how Integration Objects interact with HATS or the operating environment, always use the default templates, HPubTemplateHODBean.Default and HPubTemplateHODBeanInfo.Default. You do not need to take any action to use these templates, unless you have previously selected a different template.

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

If you use either the customizable templates or renamed copies of the templates, you must tell HATS which templates to use when creating Integration Objects. To select Integration Object templates in HATS Toolkit, click **Window > Preferences**. Expand **HATS** and click **Integration Object**. **Browse** to locate the Integration Object templates you want to use.

Choosing Integration Object templates for a bidirectional project

Two templates are provided as additional functions for HATS projects that use bidirectional code pages. These templates are:

- HPubTemplateHODBeanBIDI.Default
- HPubTemplateHODBeanBIDI.Customize

If you create a HATS project that uses a bidirectional code page, HPubTemplateHODBeanBIDI.Default is used by default when you create an Integration Object. If you wish to add customization to your Integration Objects, modify HPubTemplateHODBeanBIDI.Customize and select it on the HATS Preferences page for use in creating Integration Objects.

These templates enable you to specify whether text output by the Integration Object should be reordered. In addition, the data might be retrieved from an application working in Logical mode and displayed by an application working in Visual mode. Therefore, bidirectional reordering for insert and extract can be controlled separately. The following properties are available:

- PromptReordering - determines whether data retrieved from a Prompt action should be reordered.
- ExtractReordering - determines whether data retrieved for an Extract action should be reordered.
- PromptRTLTextOrientation - determines whether data is sent to HATS Web Services in logical right-to-left when promptReordering is `true` or, visual right-to-left format if promptReordering is `false`.
- ExtractRTLTextOrientation - determines whether data from HATS Web Services is received in logical right-to-left if promptReordering is `true` or, visual right-to-left format if promptReordering is `false`.

- PreventRoundTrip: If ExtractReordering is set to `true`, LRM markers are inserted in the data to prevent a round-trip problem and achieve correct reordering. However presence of these markers might impact data integrity, for example, if the data is needed to be sent to a system which does not process these markers properly. Setting the property to `false` turns this feature off.

The following methods are supplied to get and set these properties:

```
public String getPromptReordering();
public String getExtractReordering();
public String getPreventRoundTrip();
public void setPromptReordering(String value);
public void setExtractReordering(String value);
public void setPreventRoundTrip(String value);
```

- `public void setPromptRTLTextOrientation(String v)`

  Sets whether data retrieved from a Prompt action should be displayed in right-to-left direction (true) or not (false), where parameter *v* is either true or false

  Set parameter *v* to true if the data from a Prompt action has right-to-left direction.

- `public String getPromptRTLTextOrientation()`

  Returns whether data retrieved from a Prompt action should be displayed in right-to-left direction (true) or not (false).

- `public void setExtractRTLTextOrientation(String v)`

  Sets whether data retrieved from an Extract action should be displayed in right-to-left direction (true) or not (false), where parameter *v* is either true or false:

  Set parameter *v* to true if the data from an Extract action has right-to-left direction.

- `public String getExtractRTLTextOrientation()`

  Returns whether data retrieved from an Extract action should be displayed in right-to-left direction (true) or not (false)

- `public void setPreventRoundTrip(String v)`

  Sets whether data retrieved from an Extract action can include LRMs (true) or not (false) to achieve a correct reordering, where parameter *v* is either true or false:

  Set parameter *v* to allow a usage of these markers.

- `public String getPreventRoundTrip()`

  Returns whether usage of LRMs is allowed (true) or not (false)

where *value* can be `true` or `false`. By default these properties are set to false, so that the Integration Objects behave like those that are created with non-bidirectional templates. However, when you use the HATS Toolkit to create Model 1 or JSF Web pages, or for bottom-up Web services, that are based on Integration Objects that were created with these templates, these values are set to `true` according to options chosen in the studio GUI.

Modifying Java coding templates

The HPubTemplateHODBean.Customize and HPubTemplateHODBeanInfo.Customize templates contain Java code that is incorporated into the Integration Object Java bean code (.java) file when the Integration Object is compiled. The templates also contain constructs specifically for HATS, which are prefaced with a percent sign (%). These constructs enable

HATS to create Java beans from the data that is specified by the user when the Integration Object is created. When modifying the template files, be careful not to delete the statements containing the HATS constructs. Make backup copies of the HPubTemplateHODBean.Customize and HPubTemplateHODBeanInfo.Customize templates before you begin making changes to the template files.

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

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

Follow these steps:

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

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

   For example:

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

..
    pullVariableValueFromExtractData( haovWorkOnThis, data);

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

3. Update the HATS preferences to point to your new templates.
4. Create an Integration Object as you normally would. If you want to modify an existing Integration Object to trace the name and the screen coordinates of the Host On-Demand Extract Events, re-create the Integration Object by right-clicking on the macro and selecting **Create Integration Object**. If you introduced Java syntax errors in your template changes, they will show up as compile error messages in the task list.
5. Rebuild your HATS project. In the HATS Projects view, select the name of your project and select **Project > Clean** from the Rational SDP menu bar. You can clean all workspace projects or just selected projects.
6. If you already have a way to invoke your Integration Object, skip this step. To test your Integration Object, right click the name of the Integration Object and select either **Create Model 1 Web pages**, **Create Struts Web pages**, or **Create**

JSF Web Pages. This creates a page from which you can supply the required inputs and invoke your Integration Object.

7. Test your modified Integration Object using the Run on Server function. In the HATS Projects view, right click the name of your project and select **Run on Server**.

Sample modified Integration Object template

A sample Integration Object template, created by adding code to HPubTemplateHODBean.Customize, is contained in the HTML version of this document. To view this sample, see the HATS Knowledge Center at http://www.ibm.com/support/knowledgecenter/SSXKAY_9.5.0 and search on `Sample modified Integration Object template`.

This sample illustrates the use of a custom screen recognition criterion, which is added to the macro within the description of the appropriate screen, to trigger the DoReco() method, which is defined in the template. DoReco() saves all the fields of the host screen in an XML string named `extendedxml`. A getter method, getExtendedxml(), is provided so that the value can be extracted by a JSP after the Integration Object has been executed. The changes made to the template are marked with the comment `// ADDED FOR XML TABLE`.

The use of a custom screen recognition criterion, or descriptor, is of particular interest because it enables you to capture the content of any screen that is encountered by the Integration Object. Integration Objects are not notified when the host screen changes or when a screen is recognized. By inserting a custom screen recognition criterion, you can work with any screen.

To invoke the code that has been added to the template, you must modify the macro from which you will build the Integration Object. Locate the screen whose information you want to capture, and add this line as the last descriptor:

```
<customreco id="HPubExtractFieldAttributes|" optional="false" invertmatch="false" />
```

Be sure to add it after the other descriptors, so that it is used only on the desired screens; otherwise you might collect data from the wrong screens. Note that the customreco ID is `"HPubExtractFieldAttributes"`, but the line added to the macro has `"HPubExtractFieldAttributes|"`. The bar character (|) is used as a separator for parameters and must be included here even though there are no parameters.

If the macro uses the uselogic attribute to combine descriptors, you must update the uselogic value to include the new descriptor, or it will be ignored. Here is an example of the modifications you must make to the macro. If the original screen description is:

```
<screen name="Screen2.2" entryscreen="true" exitscreen="true" transient="false">
        <description uselogic="1 and 2" >
            <oia status="NOTINHIBITED" optional="false" invertmatch="false" />
            <string value="Ready; T" row="1" col="1" erow="-1" ecol="-1"
      casesense="true" optional="false"  invertmatch="false" />
        </description>
```

Then the screen description with the new descriptor and the modified uselogic is:

```
<screen name="Screen2.2" entryscreen="true" exitscreen="true" transient="false">
        <description uselogic="1 and (2 and 3)" >
            <oia status="NOTINHIBITED" optional="false" invertmatch="false" />
            <string value="Ready; T" row="1" col="1" erow="-1" ecol="-1"
```

```
casesense="true" optional="false" invertmatch="false" />
      <customreco id="HPubExtractFieldAttributes|" optional="false"
invertmatch="false" />
  </description>
```

When the DoReco() method is called, the template checks whether it is being called to process the "HPubExtractFieldAttributes|" custom descriptor. If so, it captures the information from the screen; if not, it passes control to the parent method to perform screen recognition using one of the other descriptors. After the Integration Object completes, the calling JSP can use the getter method to obtain the XML string and then work with it.

Extracting data from non-text planes

Extracting data from non-text planes in macros for Integration Objects is not supported by the Visual Macro Editor. However, you can extract data from non-text planes by modifying the Integration Object template.

In the example, there is a section describing a `Callback for HOD Custom screen recognition`. Using this method, other planes of data can be extracted in an IO. For example, to get color data the following code could be used in the DoReco method:

```
char[] buff = new char[2];
evt.GetPS().GetScreen(buff, 2, i, startCol, 1, ECLConstants.COLOR_PLANE);
int val = buff[0];
```

Using Integration Objects in a WebSphere Java EE application

This section describes how to use Integration Objects in two types of applications:

- A Web container, such as a custom servlet or JSP
- An EJB container

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

The instructions in this section refer to Integration Object methods. See “Integration Object methods” on page 54 for a description of these methods.

You can create your own Web project that runs Integration Objects. This section lists the steps to move files from your HATS project to another Web project and configure it to run your Integration Objects. This set of steps supports exporting and using one or more individual Integration Objects. If you want to use a chain of Integration Objects, you must copy all the files as described here and ensure that the Integration Objects run in the correct order.

HATS maintenance is not applied to Integration Objects that are deployed in a separate Web project. The best way to apply maintenance to Integration Objects used in this way is to update the Integration Objects in a HATS project and then re-export them after performing the following steps:

1. Create a Web project, if it does not already exist. This is the target project to which you are exporting the Integration Objects. Copy the Integration Object and BeanInfo source files into the **Source** directory of the Web project. Be sure to keep the IntegrationObject package.
2. Copy the profiles directory located at **Web Content/WEB-INF/profiles** into the WEB-INF directory of the new project. You can copy the entire directory, or you can re-create the directory structure and copy just the connections and macros

subdirectories of the profiles directory, with just the connection and macro that are used by the Integration Object. Either way, you need to have these files:

```
WEB-INF\profiles\application.hap
WEB-INF\profiles\connections\ioconn.hco
WEB-INF\profiles\macros\iomacro.hma
```

3. Edit the application.hap file to remove unnecessary information. Right click the file, select **Open with..** and select the text editor. When you save your changes in the text editor, you might see a message saying that you are saving a resource in a non-workbench encoding. This is because the file is UTF-8 encoded, which is required. Click **Yes** to continue. If you prefer, you can make a backup copy of application.hap before you copy it to the Web project, and edit it using the HATS editor.

   The only information application.hap must contain is the connections:

```
<?xml version="1.0" encoding="UTF-8"?>
<application active="true" configured="true" description=""
    template="Simple1.jsp">
    <connections default="ioconn">
        <connection name="ioconn"/>
   <connection name="io2conn"/>
    </connections>
</application>
```

4. Add all the jar files in the HATS EAR file into the class path of the Web project, by copying them to the WEB-INF/lib directory of the Web project.

   Note: You can also copy all the jar files in the HATS EAR file to your own EAR file. However, before doing this, you must update the project's Java Buildpath explicitly to point to the files and update the web project MANIFEST.MF to point to include those files on the CLASSPATH.

5. Copy the runtime.properties file into the same directory where you added the jar files. The location of the logs directory will be in the same directory as the runtime.properties file.

   Note: If you want to test your project in the local test environments, also copy the runtime-debug.properties file.

6. Add three function calls to initiate the HATS runtime in your servlet or JSP.

```
// Initialize and activate the HATS runtime RAS functions,
//  including tracing, logging, PII retrieval, locale.
com.ibm.hats.util.Ras.initializeRas(getServletConfig());

// Create the license manager
com.ibm.hats.util.LicenseManager.getInstance();

// Initialize Host Publisher / connection management runtime
com.ibm.hats.runtime.connmgr.Runtime.initRuntime(getServletConfig());
```

After performing these steps, you can use the Integration Object as a regular Java bean in your Web project.

To write a servlet that invokes an Integration Object:

1. Create an instance of your Integration Object by calling its constructor.
2. Invoke the setter methods for the Integration Object to set properties of input variables. The naming convention for setter methods is as follows:

   ```
   void setXyz(String)
   ```

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

You can use a different connection pool from the one you specified when you created your Integration Object. To specify a different connection pool, invoke the method specifying the name of the connection pool you want to use:

```
void setHPubStartPoolName(String)
```

3. Invoke the Integration Object to perform its task (running a macro, for example):

   ```
   void doHPTransaction(HttpServletRequest, HttpServletResponse)
   ```

4. Check for errors. The doHPTransaction(HttpServletRequest, HttpServletResponse) method throws an exception (of type com.ibm.HostPublisher.IntegrationObject.BeanException) if the Integration Object has an error.

   When the Integration Object is called by a JSP, the JSP processor catches the exception and redirects the browser to the error page that is specified on the `errorPage="errorPageName"` attribute of the `page` directive. Refer to the HATS default error page, DefaultErrorPage.jsp, for an example.

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

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

5. Request the results from your Integration Object:
   - Retrieve the values of 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 throws an ArrayIndexOutOfBoundsException exception when you have reached the end of the array.
   - Regardless of the application that you used to create your Integration Object, you can retrieve the output data in XML format by invoking the following 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.

To verify input or output variable names that are generated from data that you entered, look at the properties that are defined in your Integration Object's BeanInfo java file. The Integration Object's BeanInfo .java file is in the **Source** folder of your project in the IntegrationObject package. In HATS Toolkit, the BeanInfo file is visible only in the Navigator view.

Using an Integration Object in an EJB container (from your own EJB)

The instructions in this section refer to Integration Object methods. See "Integration Object methods" on page 54 for a description of these methods.

You can create your own EJB project that runs Integration Objects. This section lists the steps to move files from your HATS project to another EJB project and configure it to run your Integration Objects. This set of steps support exporting and using one or more individual Integration Objects. If you want to use a chain of Integration Objects, you must copy all the files as described here and ensure that the Integration Objects run in the correct order.

Notes:

1. If using chained Integration Obejcts, use a stateful EJB project. Also, you must manage the HATS session key (hPubLinkKey) in your EJB project. For more information see "Integration Object chaining" on page 59.
2. If not using chained Integration Object, use a stateless EJB project.
3. When running in a WebSphere Application Server cluster, session affinity is handled by the EJB container.

HATS maintenance is not applied to Integration Objects that are deployed in a separate EJB project. The best way to apply maintenance to Integration Objects used in this way is to update the Integration Objects in a HATS project and then re-export them after performing the following steps. In addition, if you export HATS runtime .jar files from a HATS EAR into your project, you must re-export them if you apply HATS maintenance.

1. Create a custom EJB project (**File > New > Project > EJB Project**). In this example, the project is referred to as My_EJB. Copy the application.hap file to the **ejbModule** folder of the EJB project.
2. In the **ejbModule** folder of the EJB project, create a folder named **connections** (**File > New > Other** , expand **Simple** and select **Folder**). Copy to this folder the .hco file that defines the connection that is used by the Integration Object.
3. In the **ejbModule** folder of the EJB project, create a folder named **macros**. Copy to this folder the .hma file that defines the macro that is used by the Integration Object, as well as any connect or disconnect macros that are required by your connections.
4. In the **ejbModule** folder of the EJB project, create a folder named **IntegrationObject**. Copy to this folder the Integration Object files (*.java and *BeanInfo.java). At this point you should have these files:

   ```
   ejbModule\application.hap
   ejbModule\connections\ioconn.hco
   ejbModule\macros\*.hma
   ejbModule\IntegrationObject\*.java
   ```

5. Add all the jar files that are contained in the HATS EAR file into the class path of the EJB project, by moving them either to your own EAR file, or into the **ejbModule** directory of the EJB project.

6. Copy the runtime.properties file into the same directory where you added the jar files. The location of the logs directory will be in the same directory as the runtime.properties file.

   Note: If you want to test your project in the local test environments, also copy the runtime-debug.properties file.
7. In the ejbModule\META-INF folder of the EJB project, edit the MANIFEST.MF file and add all the HATS runtime jar files to the dependency list.
8. Add these calls to your code to initiate the HATS runtime:

```
// obj is the EJB object reference,
// e.g., com.ibm.hats.util.Ras.initializeRas(this);
 com.ibm.hats.util.Ras.initializeRas(obj);
 com.ibm.hats.util.LicenseManager.getInstance();
 // appName is a String representing the EJB project
 // and obj is the EJB object reference
 // e.g. com.ibm.hats.runtime.Runtime.initRuntime(My_EJB, this);
 com.ibm.hats.runtime.connmgr.Runtime.initRuntime(appName, obj);
```

To use an Integration Object from a custom EJB:

1. Create an instance of your Integration Object by calling its constructor.
2. Invoke the setter methods 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.
3. Set the pool name to the name of the connection pool that the Integration Object will use. Note that the connection name must be qualified with the name of the EJB project. For example:

   ```
   void setHPubStartPoolName("My_EJB/main");
   ```

   Where *My_EJB* is the name of the EJB project, and main is the name of the Integration Object's connection pool.
4. Invoke the Integration Object to perform its task (running a macro, for example), using the method:

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

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

   You can reset the input variables and invoke the processRequest() method multiple times. The error indications and result values are 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. 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 *HATS Messages*. Note that the first seven characters are set to HAT*xxxx* or HPS*xxxx*, where *xxxx* is the message number.
6. Request the results from your Integration Object.

- Retrieve the values of 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 throws an ArrayIndexOutOfBoundsException exception when you have reached the end of the array.
- Regardless of the application that you used to create your Integration Object, you can retrieve the output data in XML format by invoking the following 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 that are defined in your Integration Object's BeanInfo .java source file. The Integration Object's BeanInfo .java source file is in the **Source** folder of your project in the IntegrationObject package. The BeanInfo file is visible only in the Navigator view.

Connection management API

The HATS runtime has added a new API that you can use to acquire the transformation connection (also referred to as the "default connection") in a servlet context in anticipation of executing a middle-in-chain Integration Object against that transformation connection. The purpose of this new API is to provide better integration between HATS applications and other, non-HATS Web applications. The API consists of two new static methods, acquireExistingTransformationConnection and releaseExistingTransformationConnection, added to the RuntimeFunctions class in the com.ibm.hats.runtime package.

The signatures of the method are as follows:

- `public static final String acquireExistingTransformationConnection(HttpServletRequest request)`

```
throws
HostConnectionException, ApplicationUnavailableException
```

- `public static final void releaseExistingTransformationConnection(HttpServletRequest request)`

acquireExistingTransformationConnection

The purpose of the acquireExistingTransformationConnection method is twofold:

- Allow a middle-in-chain Integration Object access to the transformation connection from a servlet context
- Check out the application from the client container to block subsequent requests from the browser while the initial request is still being processed.

This means that if a user attempts to access the transformation connection after it has been acquired and before it has been released, the browser displays an "application is busy" message page, which states that a possible cause is reloading the Web page before the application is ready. The caller uses the returned label to call the setHPubStartChainName() method of the Integration Object. The Integration Object uses this label to locate the connection against which to run. If a null label was returned, then the connection is not accessible to the Integration Object. If a HostConnectionException is thrown, the application instance exists, but there is no transformation connection. If an ApplicationUnavailableException is thrown, no application instance exists or, if it does, it is currently checked out. To prevent the Integration Object from destroying the transformation connection when an error is encountered, call the setHPubSaveConnOnError(true) method prior to invoking doHPTransaction() to run the Integration Object.

releaseExistingTransformationConnection

The purpose of the releaseExistingTransformationConnection method is to check in the application so that it can be used through the entry servlet or by other Integration Object chains. The caller should invoke the releaseExistingTransformationConnection after all Integration Objects in the Integration Object chain have been run. The releaseExistingTransformationConnection call should be used to make the transformation connection available again even if an Integration Object encounters an error while running. This call signifies that the Integration Object chain no longer needs access to the transformation connection.

Chapter 10. Creating plug-ins for Web Express Logon

The Web Express Logon feature of HATS enables you to accept network security credentials and use them to generate and use host credentials, freeing your users from the requirement to navigate host logon screens. Web Express Logon accomplishes this by the use of two types of plug-ins:

- The *Network Security plug-in* retrieves the user's credentials from a network security application.
- The *Credential Mapper plug-in* uses the credentials that are returned by the Network Security plug-in to retrieve the host user ID and acquire the host access credentials.

HATS supplies several Network Security plug-ins and Credential Mapper plug-ins. If the plug-ins that are supplied with HATS do not meet your needs, you can create your own plug-ins and integrate them into Web Express Logon. For example, if you need to access a different type of database where your credentials are stored, you can write your own plug-in.

Creating custom plug-ins for Web Express Logon

Web Express Logon relies on plug-ins to provide the network user ID and host access credentials. Web Express Logon interacts with these plug-ins through Java interfaces, which are described in the following sections.

Web Express Logon is implemented at the enterprise archive (.ear) file level for HATS Web projects and at the Web archive (.war) file level for HATS portlet projects. Your plug-in needs to be placed into each enterprise archive file or Web archive file accordingly. Follow these steps to create your own plug-in:

1. Create a Java project in Rational SDP to hold your plug-in. Ensure that these HATS files are in the class path of the Java project, in this order:
   a. hatscommon.jar
   b. hodwel.jar

   These files are located in the common plug-ins (cache) directory of the HATS Toolkit (*common plug-ins directory*/plugins/com.ibm.hats.core_x.x.x.year*xxxxxxxx*/lib) where x.x.x is the version level and year*xxxxxxxx* is the build level of HATS and in the root directory of the HATS enterprise archive file.
2. Code your plug-in using the APIs described in this chapter.
3. Create a Java Archive (.jar) file from your Java project. For HATS Web projects, export the .jar file to the file system in each .ear file in which you want to use it. For HATS portlet projects, export the .jar file to the Web Content/WEB-INF/lib directory of the HATS portlet projects in which you want to use it.
4. Add the Java Archive to the MANIFEST.MF located in Web-Content/META-INF.
5. Configure your connections to use Web Express Logon, and configure Web Express Logon to use your custom plug-in. Refer to *HATS User's and Administrator's Guide* for information about planning and configuring Web Express Logon.

Web Express Logon plug-in interface

All plug-ins must implement the following Java interfaces:

com.ibm.eNetwork.security.sso.cms.CMInterface

The CMInterface interface contains the following methods:

public int Init(Properties p, String id)
This method is used to initialize the plug-in. Any configuration parameters that are needed to initialize the plug-in are passed in with the properties object parameter. The parameters are configured in HATS Toolkit and stored in the Web Express Logon configuration file, hatswelcfg.xml. The **id** parameter is the symbolic name of the plug-in that is generated by HATS. This value is used to qualify the instance of the plug-in in the event multiple instances of the plug-in are running. The Network Security plug-in always get initialized with the default value of "" (empty string), because only one instance of this plug-in ever gets created. This method should return one of the **SSOConstants** values listed in Table 5 on page 106.

public void Destroy()
This method is called when HATS is shutting down.

public CMResponse CMSGetUserCredentials(CMRequest req)
HATS calls this method when it has selected the plug-in to respond to a request. If the plug-in is a network security type, it is expected that the plug-in will return the user's network user id. If the plug-in is a host user credential type, then this method will need to return the user's host credentials. The CMResponse and CMRequest objects used by this method are described below.

The following methods are needed for plug-in identification and selection.

public String getName();
This method returns a string that identifies the plug-in.

public String getDescription();
This method returns a string that contains information that describes the purpose and function of the plug-in.

public String getAuthor();
This method is needed to identify the originating company or person of the plug-in.

public String[] getParameters();
This method returns a string array containing the names of parameters that can be used to configure this plug-in. These tokens are the keys that are specified in the parameters section of the Web Express Logon editor in HATS Toolkit. The symbolic name of the plug-in generated by HATS (id) is prepended to each parameter name when the parameters are passed into the `Init()` method. If no parameters are needed for configuration, the method might return null.

public Properties getParameterInfo(String strParm);
Given a parameter token, this method returns a properties object with the list of properties for the given parameter. Possible properties are as follows:

- cmiDefaultValue. This property contains the default value for the specified parameter. This property is optional; if it is not specified, there is no default value for the parameter.

- cmiEncrypted. This property determines whether the parameter must be encrypted. Valid values are the strings **true** and **false**. If this parameter is set to **true**, you must use the encryption and decryption methods that are provided by Web Express Logon. See "Encrypting and decrypting plug-in parameter strings" on page 108 for information about these methods.
- cmiRequired. This property identifies whether or not a parameter is required for initialization of the plug-in. Valid values are the strings **true** and **false**.

com.ibm.eNetwork.security.sso.CMRequest

HATS requests credentials by passing request information in this object. The following are its members and methods.

Members:

- ID (Host ID or Network ID)
- Host Application ID
- Host Destination Address
- Authentication Type
- HTTP Servlet request object

Methods:

public CMRequest()

public CMRequest(String id, String applID, String hostAddr, int authType, HttpServletRequest httpRequest)

public String getID()

public void setID(String id)

public String getHostApplID()

public void setHostApplID(String applID)

public String getHostDestination()

public void setHostDestination(String hostAddr)

public int getAuthType()

Possible values for authtype are:

```
public static final int SSO_AUTHTYPE_ALL = 0xFFFF;
public static final int SSO_AUTHTYPE_3270HOST = 0x0001;
public static final int SSO_AUTHTYPE_5250HOST = 0x0002;
public static final int SSO_AUTHTYPE_VTHOST   = 0x0004;
```

These values are defined in the com.ibm.eNetwork.security.sso.SSOConstants class. You must import this class if you want to use the values.

public void setAuthType(int authType)

Possible values for authtype are:

```
public static final int SSO_AUTHTYPE_ALL = 0xFFFF;
public static final int SSO_AUTHTYPE_3270HOST = 0x0001;
public static final int SSO_AUTHTYPE_5250HOST = 0x0002;
public static final int SSO_AUTHTYPE_VTHOST   = 0x0004;
```

These values are defined in the com.ibm.eNetwork.security.sso.SSOConstants class. You must import this class if you want to use the values.

public HttpServletRequest getHttpRequestObject()

public void setHttpRequestObject(HttpServletRequest httpRequest)

com.ibm.eNetwork.security.sso.CMResponse

Your plug-in bundles its response into this object and returns it to HATS. The following are its members and methods.

Members:

- Status Code
- ID (Host ID or Network ID)
- User Credentials (Password or Passticket)

Methods:

public CMResponse()

public CMResponse(Object id, Object password, int status)

public int getStatus()

public void setStatus(int status)

The Credential Mapper plug-in uses the `status` element to provide the status of the return value. If the plug-in query fails for any reason, this field reports that failure to Web Express Logon. Failure codes are defined in the com.ibm.eNetwork.security.sso.SSOConstants class. Table 5 contains the status code numeric values, constant strings, and definitions.

Table 5. Status code definitions

Status code	Constant value	Description
0	SSO_CMR_SUCCESS	Success
1	SSO_CMR_UNKNOWN_STATUS_CODE	Unknown status code
2	SSO_CMR_CREDENTIAL_MAPPER _NOT_FOUND	Credential Mapper not found
3	SSO_CMR_INVALID_WEB_ID	Web ID not valid
4	SSO_CMR_INVALID_APPL_ID	Application ID not valid
5	SSO_CMR_INVALID_SERVER_ADDR	Server address not valid
6	SSO_CMR_DATABASE_CONNECTION_ ERROR	Database connection error
7	SSO_CMR_USER_ID_NOT_FOUND_ IN_DB	User ID not found in database
8	SSO_CMR_EXCEPTION	Exception
9	SSO_CMR_INVALID_USER_IDU	Invalid user ID not valid
10	SSO_CMR_PASSTICKET_ERROR	Passticket error
11	SSO_CMR_TIMEOUT	Timeout
12	SSO_CMR_UNEXPECTED_DCAS_RC	Unexpected DCAS return code
13	SSO_CMR_API_NOT_SUPPORTED	API not supported
14	SSO_CMR_BAD_URL	Bad URL

Table 5. Status code definitions (continued)

Status code	Constant value	Description
15	SSO_CMR_UNABLE_TO_PARSE_RESPONSE	Unable to parse response
16	SSO_CMR_LOCAL_USERID_NOT_AVAILABLE	Local user ID not available
17	SSO_CMR_DUPLICATE_XML_TAGS	Duplicate XML tags
18	SSO_CMR_CLIENT_EXCEPTION	An exception occurred while processing the credential request
19	SSO_CMR_NO_NETWORK_SECURITY_PLUGIN	Network Security plug-in is not defined to Web Express Logon

public Object getID()

public String getIDasString()

public void setID(Object id)
Your CMSGetUserCredentials() method can use this method to return the network user ID from a Network Security plug-in or the host user ID from a Credential Mapper plug-in.

public Object getPassword()

public String getPasswordasString()

public void setPassword(Object password)

Writing a Network Security plug-in

HATS provides a Network Security plug-in for Tivoli® Access Manager and one for use with WebSphere Portal. If you decide not to use these plug-ins, you can create your own.

The function of the Network Security plug-in is to acquire the user's network ID, which can be gleaned from the HTTP header that is found on the incoming HTTP request object. The details of how to acquire the network ID are specific to your network security application. Refer to your network security application's documentation for more information.

Writing a Credential Mapper plug-in

HATS provides several Credential Mapper plug-ins. If you decide not to use any of these, you can create your own plug-in.

The function of the Credential Mapper plug-in is to take the user's network ID (and perhaps the application ID) and obtain the appropriate host credentials. In Web Express Logon's implementation, users' network IDs are mapped to their host IDs during this process by way of a Java Database Connectivity (JDBC) accessible database. However, you might want to do this by another means, such as LDAP. For this reason, you might want to write your own Credential Mapper plug-in.

In the DCAS/JDBC plug-in, HATS automates z/OS® logins by associating a user's network IDs with their host IDs, and taking the host ID with the application ID and obtaining a RACF-generated passticket from the Digital Certificate Access Server (DCAS), a component of z/OS. This passticket is then used to sign the user on to the host. In your environment, you might not want to use the JDBC

association aspect of the provided plug-in. For this reason, HATS provides a DCAS API. This API provides access to RACF-generated passtickets.

Sample Web Express Logon plug-in

A sample Web Express Logon plug-in, which illustrates an approach to coding either a Network Security plug-in or a Credential Mapper plug-in, is contained in the HTML version of this document. To view this sample, see the HATS Knowledge Center at http://www.ibm.com/support/knowledgecenter/SSXKAY_9.5.0 and search on `Sample Web Express Logon plug-in`.

Encrypting and decrypting plug-in parameter strings

Web Express Logon includes an object called `com.ibm.eNetwork.security.sso.PasswordCipher` to encrypt and decrypt plug-in parameter strings. It contains the following two methods:

public static String encrypt (String plainText)
This method returns an encrypted string that is passed as a parameter.

public static String decrypt (String cipherText)
This method reverses the encryption process by returning a decrypted string. If the cipherText was not encrypted using the encrypt method, it returns the original input string

The DCAS API object

The Digital Certificate Access Server (DCAS) API object (DCASClient) encapsulates the passticket requests. The following are its members and methods.

Members:

- Port Number
- Keystore File Name
- Keystore Password
- Use WellKnownTrustedCAs
- Server Authentication
- Trace Level

If you have previously written your own plug-in using the DCASClient object prior to HATS V7.0, you do not need to change your code for it to continue to work. However, new APIs are available starting with HATS V7.0 and should be used when developing new plug-ins. The preferred APIs are flagged below with a **Preferred** symbol.

Methods:

Public DCASClient()
This constructor should be used if you want to use the default trace level and log file name when the object is created.

Public DCASClient(int *traceLevel*, String *logFile*)
This constructor should be used if you want to specify a trace level and log file name when the object is created.

- traceLevel - Trace level (0=None, 1=Minimum, 2=Normal, 3=Maximum)
- logFile - Trace log file name. This parameter is not used in HATS. Traces are recorded in the HATS trace files. Web Express Logon traces are controlled with the runtime trace flag, `trace.RUNTIME`.

public int Init (String *hostAddress*, **int** *hostPort*, **String** *trustStoreName*, **String** *trustStorePassword*, **String** *trustStoreType*)

Preferred

- *hostAddress* - The DCAS server IP address.
- *hostPort* - The DCAS server port number. If not specified, the default port number 8990 will be used.
- *trustStoreName* - The name of the truststore to be used by JSSE to connect to DCAS. It should include the full path name. The name is set to null if you are using the default truststore or WellKnownTrustedCAs.p12.
- *trustStorePassword* - The password of the specified truststore. The password is set to null if you are using the default truststore or WellKnownTrustedCAs.p12.
- *trustStoreType* - The type of the specified truststore. Valid values are:
  - DCASClient.TRUSTSTORE_TYPE_PKCS12 (pkcs12)
  - DCASClient.TRUSTSTORE_TYPE_JCEKS (jceks)
  - DCASClient.TRUSTSTORE_TYPE_JKS (jks)

  The truststore type is set to null if you are using the default truststore or WellKnownTrustedCAs.p12.

  This method should be called after creating the DCASClient object. The parameters are stored in the object, and they do not change for the life of the object. The truststore name should include the full path name. The truststore must contain the DCAS client certificate and DCAS server certificate, unless the default truststore or WellKnownTrustedCAs.p12 will be used. The truststore password should be encrypted using the encrypt password tool. It will be decrypted before being stored in the object. The valid return codes are described in the SSOConstants object. Return 0 on success, nonzero otherwise. See SSOConstants for return codes.

Public int Init(String *dcasAddr*, **int** *dcasPort*, **String** *keystoreFileName*, **String** *keystorePassword*)

This method should be called after creating the DCASClient object. The parameters are stored in the object, and they do not change for the life of the object. The *keystoreFileName* should include the full path name. The keystore database must contain a DCAS client certificate and the DCAS server certificate. If WellKnownTrustedCAs.p12 or the default truststore is being used for these certificates, set *keystoreFileName* and *keystorePassword* to null. The keystore password should be encrypted using the encrypt password tool. It will be decrypted before being stored in the object. The valid return codes are described in the SSOConstants object.

If a keystore of a type other than p12 is being used, use the `Init(hostAddress, hostPort, trustStoreName, trustStorePassword, trustStoreType` method instead.

- *dcasAddr* - The DCAS server's IP address
- *dcasPort* - The DCAS's port number. If not specified, the default port number of 8990 will be used.
- *keystoreFileName* - The name of the SSL keystore database file. The name should include the full path name. While you are developing your plug-in, you will not know the full path that will be used when you deploy your HATS application. Following is an example of code that

you can use to convert a provided file name parameter into a fully qualified file name, relative to the EAR directory, at runtime:

```
import com.ibm.eNetwork.security.sso.cms.CredMapper;
import com.ibm.eNetwork.security.sso.cms.PluginResourceLocator;

    if ( (p12FileName != null) &&
     (CredMapper.getPluginResourceLocator() != null))
        p12FileName =
     CredMapper.getPluginResourceLocator().findResource(p12FileName);
```

- *keystorePassword* - The password of the above keystore database.

The keystore Password should be encrypted with the `PasswordCipher.encrypt()` method. If it is provided by an encrypted parameter (that is, a parameter with **cmiEncrypted="true"**, it is provided to the plug-in in encrypted form. If it is not provided by an encrypted parameter, you can use this code to encrypt it:

```
import com.ibm.eNetwork.HOD.common.PasswordCipher;
keystorePW = PasswordCipher.encrypt(keystorePW);
```

The valid return codes are listed in Table 5 on page 106.

public void setUseDefaultTrustStore(boolean def)

public void setNoFIPS(boolean nof)

Public void setWellKnownTrustedCAs(boolean wellKnownCAs)

Public void setServerAuthentication(boolean serverAuth)

Public CMResponse getPassticket(String *hostUserID*, **String** *hostApplID*, **String** *hostAddr*, **long** *timeout*)

This method should be called after creating and initializing the DCASClient object to obtain a passticket from the DCAS server. The passticket and the user ID are returned in a CMResponse object. The caller should check the status field of the CMResponse object to see whether the call was successful or not. If the call was successful, the status field will be set to SSO_CMR_SUCCESS. The valid values for the status field are listed in Table 5 on page 106. An SSL client-authenticated connection is established with the DCAS, and it is reused for all subsequent passticket requests.

- *hostUserID* - User ID for which the passticket is being requested.
- *hostApplID* - Application ID for which the passticket is being requested.
- *hostAddr* - The DCAS's address.
- *timeout* - The time available for the DCAS protocol to return a passticket, specified in milliseconds.

Public void Destroy()

This method closes the DCAS connection.

Chapter 11. Using the HATS bidirectional API

If you create HATS applications that use bidirectional (Arabic or Hebrew) code pages, and you add business logic or create your own custom components or widgets, you can use the bidirectional API to handle the recognition of host components and the presentation of widgets on the Web page. This chapter describes this API. Before using the material in this chapter you should be familiar with the bidirectional concepts described in *HATS User's and Administrator's Guide*.

Data Conversion APIs

Two APIs for handling text conversion from visual to logical and vice versa are included in the HostScreen class. You can use these APIs when creating custom widgets and components to handle the extraction of data.

ConvertVisualToLogical

```
public java.lang.String ConvertVisualToLogical(java.lang.String
inputBuffer, boolean isleft-to-rightVisual,
boolean isleft-to-rightImplicit)
```

Converts the given string from visual to implicit format and returns the implicit format of the string.

inputBuffer
The input string in visual format.

isleft-to-rightVisual
If true, inputBuffer is in visual left-to-right form.

isleft-to-rightImplicit
If true, the output buffer is in implicit left-to-right form.

ConvertLogicalToVisual

```
public java.lang.String ConvertLogicalToVisual(java.lang.String
inputBuffer, boolean isleft-to-rightImplicit,
boolean isleft-to-rightVisual)
```

Converts the given string from implicit to visual format and returns the visual format of the string.

inputBuffer
The input string in implicit format.

isleft-to-rightImplicit
If true, inputBuffer is in implicit left-to-right form.

isleft-to-rightVisual
If true, the output buffer is in visual left-to-right form.

Global Variable APIs

There are two getter methods that you can use to get the value of global variables. Using these methods you can get the global variable value either in implicit format or in visual format. These two methods are in class com.ibm.hats.common.BaseInfo.

getGlobalVariable

```
public IGlobalVariable getGlobalVariable(String name, boolean
createIfNotExist,boolean bidiImplicit)
```

Gets the named global variable, optionally creating it if it does not already exist.

createIfNotExist
Indicates whether or not to create a nonexistent global variable.

bidiImplicit
Indicates whether to get the global variable value in implicit format if `true`, or in visual format if `false`.

getSharedGlobalVariable

```
public IGlobalVariable getSharedGlobalVariable(String name, boolean
createIfNotExist,boolean bidiImplicit)
```

Gets the named shared global variable, optionally creating it if it does not already exist.

createIfNotExist
Indicates whether or not to create a nonexistent global variable

bidiImplicit
Indicates whether to get the global variable value in implicit format if `true`, or in visual format if `false`.

BIDI OrderBean

You can use the methods of the BIDI OrderBean for the correct display of bidirectional data. It contains the following parameters:

BidiString
String. Contains bidirectional text

FromTextVisual
Boolean. Indicates whether the source bidirectional text is visual. Default is `true`.

FromOriLTR
Boolean. Indicates whether the orientation of the source bidirectional text is LTR. Default is `true`.

ToTextVisual
Boolean. Indicates whether the target bidirectional text is visual. Default is `true`.

ToOriLTR
Boolean. Indicates whether the orientation of the target bidirectional text is LTR. Default is `true`.

NeedShape
Boolean. Indicates whether bidirectional text is Arabic text and whether it needs shaping. Default is `false`.

CharSet
String. Defines the character encoding for the JSP.

NumShape
String. Defines the numerals shaping method. Default is `Nominal`.

SymSwap
Boolean. Indicates whether symmetric swapping is on. Default is `false`.

BIDI OrderBean methods

setBidiString
`public void setBidiString (String BdString)`

Sets the bidirectional text to be reordered to the given string. The only parameter is **BdString**, which is the bidirectional string that needs reordering.

getBidiString
`public String getBidiString ()`

Gets the bidirectional text. Returns the bidirectional string that needs reordering.

setFromTextVisual
`public void setFromTextVisual (boolean on)`

Sets the source bidirectional text type as visual. The only parameter is **on**. If `true`, defines this source bidirectional text as visual. If `false`, defines this source bidirectional text as implicit.

setFromOriLTR
`public void setfromOriLTR (boolean on)`

Sets the source bidirectional text orientation as LTR. The only parameter is **on**. If `true`, defines this source bidirectional text as LTR. If `false`, defines this source bidirectional text as RTL.

setToTextVisual
`public void setToTextVisual (boolean on)`

Sets the target bidirectional text type as visual. The only parameter is **on**. If `true`, defines this target bidirectional text as visual. If `false`, defines this target bidirectional text as implicit.

setToOriLTR
`public void setToOriLTR (boolean on)`

Sets the target bidirectional text orientation as LTR. The only parameter is **on**. If `true`, defines this target bidirectional text as LTR. If `false`, defines this target bidirectional text as RTL.

setEncoding
`public void setEncoding (String CharSet)`

Sets the encoding character set. The only parameter is **CharSet**, which is a character-encoding name.

setNeedShape
`public void setNeedShape (boolean on)`

Sets the need to perform shaping. The only parameter is **on**. If `true`, indicates the need to perform shaping on the bidirectional text.

Order `public void Order ()`

Performs the ordering of the bidirectional text. There are no parameters.

CompressLamAlef
`public String CompressLamAlef(String input,boolean direction)`

Returns a string in which a Lam character followed by an Alef character is replaced by one LamAlef character. Parameters are:

- Direction. If `true`, indicates input text is in visual form. If `false`, input text is in implicit form.
- Input. An input string containing LamAlef characters to be compressed.

ExpandLamAlef

```
public String ExpandLamAlef(String input,boolean direction)
```

Returns a string in which a Lam Alef character is replaced by a Lam followed by one Alef character. Parameters are:

- Direction. If `true`, indicates input text is in visual form. If `false`, input text is in implicit form.
- Input. An input string containing LamAlef characters to be expanded.

setNumerals

```
public void setNumerals(String NumShape)
```

Sets the numerals shape of the output buffer. The only parameter is:

- NumShape. A string that takes one of three values:
  - NOMINAL. Numerals are in Latin format.
  - CONTEXTUAL. Numerals follow numbers.
  - NATIONAL. Numerals are in National format.
- Input. An input string containing LamAlef characters to be expanded.

setSymSwap

```
public void setSymSwap (boolean on)
```

Sets the Symmetric swapping option with Visual RTL orientation. The only parameter is **on**. If `true`, symmetric swapping is enabled for swapping characters in RTL screens. If `false` (the default), symmetric swapping is disabled for swapping characters in RTL screens.

ShapeArabicData

```
public String ShapeArabicData(String strInBuffer,boolean
isLTRVisual, boolean EnableNumSwap)
```

Returns a string in which Arabic data is shaped. Parameters are:

- strInBuffer. The bidirectional string that needs shaping.
- isLTRVisual. An input string containing LamAlef characters to be expanded. If `true`, bidirectional string is left to right visual. If `false`, bidirectional string is right to left visual.
- EnableNumSwap. If `true`, enable Numeric swapping. If `false`, disable numeric swapping.

DeshapeArabicData

```
public String DeshapeArabicData (String strInBuffer,boolean
isLTRVisual,boolean EnableNumSwap)
```

Returns a string in which Arabic data is deshaped. Parameters are:

- strInBuffer. The bidirectional string that needs deshaping.
- isLTRVisual. If `true`, bidirectional string is left to right visual. If `false`, bidirectional string is right to left visual.
- EnableNumSwap. If `true`, enable numeric swapping. If `false`, disable numeric swapping.

ConvertLogicalToVisual

```
public java.lang.String ConvertLogicalToVisual(java.lang.String
inputBuffer, boolean isLTRimplicit,
boolean isLTRVisual)
```

Converts the given string from implicit to visual format and returns the visual format of the string. Parameters are:

- InputBuffer. The input string in implicit format.
- isLTRimplicit. If `true`, inputBuffer is in implicit left-to-right form.
- isLTRVisual. If `true`, the output buffer is in visual left-to-right form.

ConvertVisualToLogical

```
public java.lang.String ConvertVisualToLogical(java.lang.String
inputBuffer, boolean isLTRVisual,
boolean isLTRimplicit)
```

Converts the given string from visual to implicit format and returns the implicit format of the string. Parameters are:

- InputBuffer. The input string in visual format.
- isLTRimplicit. If `true`, the output buffer is in implicit left-to-right form.
- isLTRVisual. If `true`, inputBuffer is in visual left-to-right form.

Appendix A. HATS Toolkit files

When you use HATS Toolkit to build your project, files for each component of the project are created. This appendix tells you where the file is located on your system, how to view and edit the source for the file, and describes the tags that make up each file.

Note: Use the HATS Toolkit editors if you edit these source files.

All of the files you create with HATS Toolkit are stored on your system in one or more workspaces managed by your Rational SDP program, such as Rational Application Developer. You can choose your workspace directory, and you can have more than one. Refer to the information provided with Rational SDP for information about choosing your workspace.

All of the file locations in this appendix refer to the relative path from the directory named for your project, which will be created within your workspace.

Note: The files contained in a HATS EJB (Enterprise JavaBeans) project are different from those in a HATS project. For a description of the contents of a HATS EJB project, see Chapter 8, "Creating and using a HATS EJB application," on page 81

Application file (.hap)

The application file contains XML tags that define the settings you select when you create the project.

The application (.hap) file is stored in the *project_name*/Web Content/WEB-INF/profiles directory, where *project_name* is the name you gave the project when you created it. The application (.hap) file for a HATS EJB project is stored in the *project_name*/ejbModule directory. To view and edit the source of the application file for your HATS project, expand your project in the **HATS Projects** view and double-click **Project Settings** to open the project editor. You can view the source by clicking on the **Source** tab.

You can modify the application file using any of the tabs in the project editor. HATS Toolkit updates the affected information on other tabs when you make changes on any tab.

<application> tag

The <application> tag is the enclosing tag for the project.

The attributes of the <application> tag are:

active This attribute is not used by HATS. It is contained here for compatibility with HATS Limited Edition.

configured
This attribute is not used by HATS. It is contained here for compatibility with HATS Limited Edition.

description
Specifies the description you enter when you create a project.

template
Specifies the name of the template you selected for the project when you created the project. The default template is Finance.jsp.

<connections> tag

The <connections> tag is a container for all the connection tags that define connections for this project.

The attributes of the <connections> tag are:

default
Specifies the name of the default connection. The default connection, which is created using the connection values that you specify in the New HATS Project wizard, defaults to the name of main.

<connection> tag

The <connection> tag identifies a connection defined for the project and points to the connection (.hco) file that defines the connection.

The attributes of the <connection> tag are:

name Specifies the name you entered when you created the connection.

<eventPriority> tag

The <eventPriority> tag is the enclosing tag for the screen events you defined for the project. The order of the event tags within the <eventPriority> tag is the order in which screen events are checked when a new host screen is encountered. This tag has no attributes.

<event> tag

The <event> tag specifies a screen event that you defined for the project.

The attributes of the <event> tag are:

enabled
Specifies whether the screen event's screen recognition criteria should be checked when a new host screen is encountered. Valid values are `true` and `false`. The default value is `true`.

name Specifies the name you gave the screen event when you defined it. If you store a screen event file under a folder (or group), the name of the folder is prepended to the name of the file.

type Specifies that this is a screen combination event. The available attribute is **screenCombination**.

<classSettings> tag

The <classSettings> tag is the enclosing tag for the Java classes you include in the project. This tag has no attributes.

<class> tag

The <class> tag specifies a class whose attributes are defined in the enclosed <setting> tags.

The attributes of the <class> tag are:

name Specifies one of the following Java classes:

- com.ibm.hats.common.AppletSettings
- com.ibm.hats.common.ApplicationKeypadTag
- com.ibm.hats.common.ClientLocale
- com.ibm.hats.common.DBCSSettings
- com.ibm.hats.common.DefaultConnectionOverrides
- com.ibm.hats.common.DefaultGVOverrides
- com.ibm.hats.common.HostKeypadTag
- com.ibm.hats.common.KeyboardSupport
- com.ibm.hats.common.OIA
- com.ibm.hats.common.RuntimeSettings
- com.ibm.hats.transform
- com.ibm.hats.transform.components.*name*
- com.ibm.hats.transform.DefaultRendering
- com.ibm.hats.transform.widgets.dojo.*name*
- com.ibm.hats.transform.widgets.*name*

<setting> tag

The <setting> tag specifies the settings associated with the class in which the <setting> tag is enclosed. The <setting> tag contains **name** and **value** pairs for each of the classes. The following sections described the **name** and **value** pairs for each of the classes.

com.ibm.hats.common.AppletSettings

For the com.ibm.hats.common.AppletSettings class, name specifies a customizable setting for the HATS application automatic disconnect and refresh implementation methods, Client pull (AJAX) and Server push (applet). The following settings can be used to configure the Client pull (AJAX) method, including the auto-disconnect and auto-refresh functions. These settings are supported for HATS Web applications, including JSR 286 portlets.

browserDisconnectDelay
Effective when the **enable** setting is `ajax` and the **browserDisconnectEnabled** setting is `true`. The time in milliseconds to wait before performing the auto-disconnect function. The minimum value is 2000 milliseconds (2 seconds). The default is **15000** milliseconds (15 seconds).

browserDisconnectEnabled
If the **enable** setting is `ajax`, and if this setting is `true`, enables the auto-disconnect function. If enabled, the HATS application initiates a disconnect action if the client has not polled the HATS application within the time specified by the **browserDisconnectDelay** setting.

browserPollInterval
Effective when the **enable** setting is `ajax`. The interval in milliseconds at which the browser will poll the HATS application to restart the **browserDisconnectDelay** timer, if enabled, and check for host screen updates. The minimum value is 1000 milliseconds (1 second). If **browserDisconnectEnabled** is `true`, then the poll interval value must be less than the value in the **browserDisconnectDelay** setting by at least 1000 milliseconds (1 second). The default is **5000** milliseconds (5 seconds).

Note: If an HTTP session idle timeout is configured, browser polling of the HATS application effectively disables the HTTP session idle timeout functionality. Because of this, the HATS runtime takes responsibility for monitoring the HTTP session idle timeout period and initiates a disconnect of the HATS session when no user activity is seen before the idle time is exceeded.

browserRefreshEnabled
If the **enable** setting is `ajax`, and if this setting is `true`, enables the auto-refresh function. If enabled, the browser initiates a refresh of the screen if there has been no user input and the poll response indicates that the host screen has changed.

enable
Use this setting to configure which automatic disconnect and refresh method to use. Specify `ajax` to configure the Client pull (AJAX) methods, auto-disconnect and auto-refresh. Specify `true` to configure the Server push (applet) method, also known as the asynchronous applet update method. Specify `false` to disable both of the methods. The default is **false**.

Refer to the section, *Using the Server push (applet) method*, in the *HATS User's and Administrator's Guide* for a description of the settings for the Server push (applet) method.

Note: The Server push (applet) method of detecting disconnect and host refresh is deprecated. You are encouraged to use the Client pull (AJAX) method instead. While the applet method is currently supported, IBM reserves the right to remove this capability in a future release of the product.

com.ibm.hats.common.ApplicationKeypadTag

For the com.ibm.hats.common.ApplicationKeypadTag class, name specifies a customizable setting:

show If `true`, shows a keypad in the application.

showDefault
If `true`, shows a key in the application keypad to change the presentation to the default transformation.

showDisconnect
If `true`, shows a key in the application keypad to disconnect from the host.

showKeyboardToggle
If `true`, shows a key in the application keypad for toggling display of a host keyboard.

showPrintJobs
If `true`, shows a key in the application keypad for showing print jobs.

showRefresh
If `true`, shows a key in the application keypad to refresh the browser window contents using the original transformation, and restore the input fields to their original value.

showReset
If `true`, shows a key in the application keypad to clear all the fields on the browser page of any entries made by the end user.

showReverse
If `true`, shows a key in the application keypad for bidirectional support.

style Depending on the value attribute, shows the keys defined with value=`buttons` or `links` in the application keypad.

com.ibm.hats.common.ClientLocale

For the com.ibm.hats.common.ClientLocale class, name is always `locale`. The value for the locale setting specifies the language to be used to display button captions and messages. Value can be one of the following. The default is **accept-language**.

Characters that identify the country code of the locale

ar Arabic

cs Czech

de German

en English

es Spanish

fr French

hu Hungarian

it Italian

ja Japanese

ko Korean

pl Polish

pt_BR Brazilian Portuguese

ru Russian

tr Turkish

zh Simplified Chinese

zh_TW
Traditional Chinese

accept-language
The language is acquired from the `Accept-Language` HTTP header of the user's browser.

primaryLocale
The language is acquired from the primary locale of the WebSphere Application Server where the HATS application runs.

com.ibm.hats.common.DBCSSettings

For the com.ibm.hats.common.DBCSSettings class, there are three settings, autoConvertSBCStoDBCS, eliminateMaxlengthInIdeographic, and showUnprotectedSISOSpace.

- Valid values for the **autoConvertSDBCtoDBCS** attribute are:

  true Automatically convert single byte characters to double byte characters for 3270 and 3270E G-type or 5250 G-type and J-type fields.

  false Do not automatically convert single byte characters to double byte characters for 3270 and 3270E G-type or 5250 G-type and J-type fields.

  The default is **false**. For more information, see the section Project settings editor in the *HATS User's and Administrator's Guide*.

- Valid values for the **eliminateMaxlengthInIdeographic** attribute are:

true If the **enableAutoAdvance** Runtime setting is `true`, then characters that exceed the maximum length of the DBCS field can be entered into the IME. Also, when the user selects the IME candidate for the field, the excess characters are cut and pasted into the next field.

If the **enableAutoAdvance** Runtime setting is `false`, then characters that exceed the length of the DBCS field can be entered into the IME. However, when the user selects the IME candidate for the field, the excess characters are removed and not entered into any other field.

false Characters that exceed the maximum length of the DBCS field cannot be entered into the IME.

The default is **false**. For more information, see the section Project settings editor in the *HATS User's and Administrator's Guide*.

- Valid values for the **showUnprotectedSISOSpace** attribute are:

  true Show any unprotected Shift In or Shift Out characters as a space.

  false Do not use a space to show unprotected Shift In or Shift Out characters.

  The default is **true**. For more information, see the section Project settings editor in the *HATS User's and Administrator's Guide*.

com.ibm.hats.common.DefaultConnectionOverrides

For the com.ibm.hats.common.DefaultConnectionOverrides class, there is always at least one <setting> tag with a name attribute of `allowAll`. This <setting> tag indicates the chosen default security policy regarding the overriding of connection parameters. Any exceptions to the chosen security policy for connection overrides are recorded with additional <setting> tags, with the name attribute set to the name of the exceptional connection parameter.

Valid values for the name attributes are:

true The end user can override the named connection parameter. If the named connection parameter is `allowAll`, this means that all unnamed connection parameters may be overridden with clients requests.

false The end user can not override the named connection parameter. If the named connection parameter is `allowAll`, this means that no unnamed connection parameters may be overridden

The default for the `allowAll` setting is **false**.

com.ibm.hats.common.DefaultGVOverrides

For the com.ibm.hats.common.DefaultGVOverrides class, there is always at least one <setting> tag with a name attribute of `allowAll`. This <setting> tag indicates the chosen default security policy regarding the overriding of global variables. Any exceptions to the chosen security policy are recorded with additional <setting> tags, with the name attribute set to hatsgv_*variableName* for regular global variable exceptions, or hatssharedgv_*variableName* for shared global variable exceptions.

Valid values for the name attributes are:

true The end user can override the named connection parameter. If the named connection parameter is `allowAll`, this means that all unnamed connection parameters may be overridden with clients requests.

false The end user can not override the named connection parameter. If the

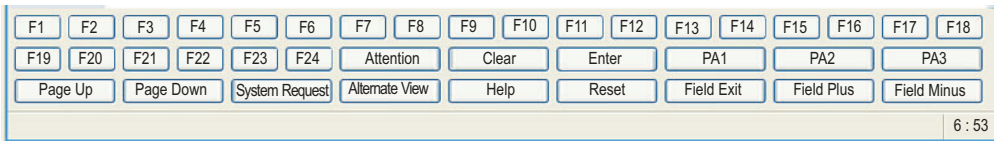
named connection parameter is `allowAll`, this means that no unnamed connection parameters may be overridden

The default for the `allowAll` setting is **false**.

com.ibm.hats.common.HostKeypadTag

For the com.ibm.hats.common.HostKeypadTag class, name specifies a customizable setting:

show If `true`, shows a host keypad in the application.



showAltView
If `true`, shows an AltView key in the host keypad.

showAttention
If `true`, shows an ATTN key in the host keypad.

showClear
If `true`, shows a CLEAR key in the host keypad.

showEnter
If `true`, shows an Enter key in the host keypad.

showF1 – showF24
If `true`, shows a Function key with the corresponding number in the host keypad.

showFieldExit
If `true`, shows a Field Exit key in the host keypad.

showFieldMinus
If `true`, shows a Field Minus key in the host keypad.

showFieldPlus
If `true`, shows a Field Plus key in the host keypad.

showHelp
If `true`, shows a Help key in the host keypad.

showPA1
If `true`, shows a PA1 key in the host keypad.

showPA2
If `true`, shows a PA2 key in the host keypad.

showPA3
If `true`, shows a PA3 key in the host keypad.

showPageDown
If `true`, shows a Page Down key in the host keypad.

showPageUp
If `true`, shows a Page Up key in the host keypad.

showPrint
If `true`, shows a PRINT key in the host keypad for printing output.

showReset
If `true`, shows a RESET key in the host keypad.

showSystemRequest
If `true`, shows a SYSREQ key in the host keypad.

style Specifies how keys defined with value=`true` are displayed in the host keypad. Valid values are `buttons` or `links`. The default is **buttons**.

com.ibm.hats.common.KeyboardSupport

For the com.ibm.hats.common.KeyboardSupport class, name specifies a customizable setting:

enable
Specifies whether keyboard support is available in the project. When keyboard support is enabled, end users can use the physical keyboard keys to interact with the host. The end user can press certain physical keys that have been mapped to host aid keys, such as the F1, SYSREQ, RESET, or ATTN keys. The end user can toggle keyboard support to be disabled if he wants to use a mapped physical keyboard key to interact with the browser.

Note: This must be set to `true` to turn on the wizard that allows the HATS theme to change from the default emulator style to a modern application style.

initialState
If `true`, the initial state of the host keyboard is on (the user can interact with the application using the physical keyboard).

supportAllKeys
If `true`, all mapped keys are supported, regardless of what buttons or links are displayed. If `false`:

- If there are no recognized host functions displayed in the current page as buttons or links, support all mapped host functions.
- If there are any recognized host function buttons or links, support only those host functions.

com.ibm.hats.common.OIA

For the com.ibm.hats.common.OIA class, name specifies a customizable setting:

active If `true`, an operator information area (OIA) is visible in the project. The default is **true**.

Note: This must be set to `true` to turn on the wizard that allows the HATS theme to change from the default emulator style or to a modern application style.

appletActive
If `true`, an indicator is displayed in the OIA if asynchronous update support is enabled. The default is **false**.

autoAdvanceIndicator
If `true`, displays in the OIA whether auto-advance is enabled, if it is supported by the browser. The default is **false**.

bidiControls
If `true`, displays in the OIA the current bidirectional controls to indicate editing status, if they are supported by the browser. The default is **true**.

cssClass
Specifies the cascading stylesheet (CSS) class name that controls the appearance of the OIA. The default is **statusArea**.

cursorPosition
If `true`, displays in the OIA the absolute cursor position for the host, such as 1391. The default is **false**.

cursorRowColumn
If `true`, displays in the OIA the row and column of the host cursor, such as 18/031. The default is **true**.

fieldData
If `true`, displays in the OIA field extended data, such as numeric only or field exit required. The default is `false`.

inputInhibited
If `true`, displays in the OIA whether the keyboard is locked, preventing input from the keyboard. The default is **true**.

insertMode
If `true`, displays in the OIA whether overwrite mode is enabled, if it is supported by the browser. The default is **true**.

layout Depending on the value attribute, determines how to display the OIA, either horizontally (across the bottom of the page) or vertically (as a side frame on the page). The default is **horizontal**.

msgWaiting
If true, displays an indicator when the host system has one or more message for the session. The setting is applicable only for 5250 host systems.

sslCheck
If `true`, displays in the OIA whether the Host On-Demand connection is SSL secured. The default is **true**.

systemWait
If `true`, displays in the OIA whether the system is locked while waiting for data to be returned. The default is **true**.

com.ibm.hats.common.RuntimeSettings

For the com.ibm.hats.common.RuntimeSettings class, name specifies a customizable setting:

autoEraseFields
Specifies whether modified input fields should have `[erasefld]` applied before modified data is entered into the field. The default value is **true**. If the value is set to `false`, space characters may be used to replace data already entered in the field by the host.

Notes:

1. Any host field that is rendered as multiple input fields will not be automatically cleared. For example, long host fields that wrap from one line to the next are rendered as multiple input fields and will not be automatically cleared before updating.
2. This setting can only be specified at the project level. It cannot be specified for a single transformation.

enableAutoAdvance
Specifies whether the cursor moves to the next input field when located at the end of an input field; that is, when the input field is entirely filled in. When `true`, the cursor will move to the next input field when located at the end of an input field. When `false`, the cursor does not move to the next input field unless the user explicitly moves it. The default is **false**.

enableAutoTabOn
Specifies whether the tab key will move the cursor to the next input field when the cursor reaches the end of an input field; that is the input field is entirely filled in. When set to `true`, based on the order of presentation field in the browser, the tab key will move cursor in the current field to the next field when the position of the cursor is at the end of the current field. When set to `false`, the tab key does not move to the next input field unless the user explicitly moves it. The default is **false**.

enableBusyPage
When set to `true`, sending multiple requests will be redirected to busy page behavior. When set to `false`, sending multiple requests will be blocked in the client side and busy page is not displayed. The default is **false**.

enableCompression
When set to `true`, enables the HTTP compression filter used to reduce the number of bytes transferred between the HATS runtime, which runs on the WebSphere Application Server, and the user's browser. The default is **false**.

escapeHTMLTags
Specifies whether HTML tag interpretation should be performed for data on a host screen. If `true`, then all data is interpreted as simple text. If `false`, then data that looks like a valid HTML tag, is treated as such. For example, if a screen contains data such as, `<S>elect`, and this setting is `true`, the data is interpreted as the simple text, `<S>elect`. If this setting is `false`, the `<S>` is interpreted as the HTML strikeout tag. The default is **false**.

Note: If `true`, data that matches a valid HTML tag, supplied using the text replacement function, is treated as simple data.

enableOverwriteMode
If `true`, text entered into an input field overwrites text at the cursor position one character at a time. If `false`, text entered into an input field is inserted at the cursor position pushing existing text ahead. The user can toggle from this initial setting using the Insert key. The default is **true**.

nextFieldForDropDown
If `true`, the cursor position is moved to the next input field when a selection is made from a drop-down list. The default for new projects created in HATS V7.5.0.2, or later, is **true**. The default for projects created before HATS V7.5.0.2 is **false**.

Note: This setting is effective only when **enableAutoAdvance** is `true`.

selectAllOnFocus
If `true`, all text in a field is selected when the field receives focus, which is typical behavior for a Web application. If `false`, no text is selected when the field receives focus which is typical behavior for a terminal emulator.

Notes:

1. For Web applications:
   - The default is **true**.
   - This setting does not affect the **enableOverwriteMode** setting behavior.
   - This setting is only valid when Internet Explorer is used as the browser for the application.

2. For rich client applications:
   - The default is **false**.
   - When selected, this setting functions like the **enableOverwriteMode** setting in that characters are overwritten as a user types into the field.
   - Text is selected only when the keyboard is used to tab into the field. Text is not selected when clicking the mouse in the field.

suppressUnchangedData
If `true`, disables all fields whose contents are the same as when the form was rendered. If `false`, sends any field contents received from the browser to the host even if the current presentation space contents are identical for that field. The default is **false**.

com.ibm.hats.transform
For the com.ibm.hats.transform class, name specifies a customizable setting:

alternate
The value `DEFAULT` if an alternateRenderingSet is specified. Otherwise, unspecified.

alternateRenderingSet
Specifies the name of the rendering set to use for default rendering if nothing is found to render during transformation of a HATS component tag.

com.ibm.hats.transform.components.*name*
Refer to the *HATS User's and Administrator's Guide* for descriptions of component settings.

com.ibm.hats.transform.DefaultRendering
For the com.ibm.hats.transform.DefaultRendering class, name is always `applicationDefaultRenderingSetName`. The value specifies the name of the rendering set defined as the default rendering set for the project. The rendering set name specified on the value setting must match the value of the default attribute specified for the <defaultRendering> tag.

com.ibm.hats.transform.widgets.dojo.*name*
Refer to the *HATS User's and Administrator's Guide* for descriptions of HATS Dojo widget settings.

com.ibm.hats.transform.widgets.*name*
Refer to the *HATS User's and Administrator's Guide* for descriptions of widget settings.

<textReplacement> tag
The <textReplacement> tag is the enclosing tag for any text replacement values you define in the project. This tag has no attributes.

<replace> tag
The <replace> tag specifies the text replacement values in a project.

Note: If you are using a bidirectional code page, refer to *HATS User's and Administrator's Guide*.

The attributes of the <replace> tag are:

caseSensitive
Specifies whether the case of text replacement values must match before text replacement occurs. Valid values are `true` and `false`. The default is `false`.

from Specifies the text you want to replace. The text on the **from** attribute must be enclosed in quotes.

to When replacing text with text or HTML coding (Web only), specifies the replacement string you want to insert in place of the value specified on the **from** attribute. The replacement string on the **to** attribute must be enclosed in quotes. If you want to replace the text with a button or a link, the code for the button or link must be added inside the quotes.

regularExpression
Specifies whether Java regular expression support is used as part of the text replacement algorithm. A regular expression is a pattern of characters that describes a set of strings. You can use regular expressions to find and modify occurrences of a pattern. Valid values are `true` and `false`. The default is `false`.

toImage
When replacing text with an image, specifies the path and name of the image you want to insert in place of the value specified on the **from** attribute. The path and name of the image on the **toImage** attribute must be enclosed in quotes.

matchLTR
When using a bidirectional code page, specifies whether the value specified on the **from** attribute is replaced on left-to-right screens. Valid values are `true` and `false`. The default is `true`.

matchRTL
When using a bidirectional code page, specifies whether the value specified on the **from** attribute is replaced on right-to-left screens. Valid values are `true` and `false`. The default is `false`.

matchReverse
When using a bidirectional code page, specifies whether the value specified on the **from** attribute is replaced when the section of the screen in which it appears has been reversed from the original direction of the page. Valid values are `true` and `false`. The default is `false`.

Note: Care should be taken when using text replacement. Text replacement with a disparate number of characters in the strings can cause changes in the representation of the screen. Depending on the widget used for presenting a region of a screen, text on a line of the screen can be contracted, expanded, or forced to a new line.

<defaultRendering> tag

The <defaultRendering> tag is the enclosing tag for all rendering sets you define in the project.

The attribute of the <defaultRendering> tag is:

default
Specifies the name of the rendering set to use for default rendering in the project. The rendering set name specified on the default attribute *must*

match the value specified for the value attribute of the class setting named com.ibm.hats.transform.DefaultRendering.

<renderingSet> tag

The <renderingSet> tag is the enclosing tag for rendering items defined in the rendering set.

The attributes of the <renderingSet> tag are:

name The name specified for the rendering set when it was created.

description
The description specified for the rendering set when it was created.

layout Indicates whether to use compact rendering, which eliminates unnecessary blanks in fields and text on the transformed screen. This attribute should only be used if you want your default rendering to be compacted. The only valid value for layout is COMPACT. By default, a rendering set does not specify this attribute and does not use compact rendering.

separated
Indicates whether to render the output using inline span tags to differentiate between fields and reduce the amount of HTML and blank space on the transformed screen. This is the default for Web applications optimized for mobile devices. By default, a rendering set does not specify this attribute.

table Indicates whether to render the output in a table and preserve the layout of the original host screen. This is the default for Web applications not optimized for mobile devices.

<renderingItem> tag

The <renderingItem> tag is the enclosing tag for a specific rendering item.

The attributes of the <renderingItem> tag are:

componentIdentifier
The name of the rendering item used to coordinate component information with the transformation. The default setting is the name of the screen combination event.

associatedScreen
The name of the captured screen used to create this rendering item.

description
The description entered when the rendering item was created.

enabled
Indicates whether this rendering item is enabled. Reflects the state of the check box on the Rendering page of Project Settings.

endCol
The last column of the host screen to which this rendering item should be applied. -1 means the rightmost column of the host screen.

endRow
The last row of the host screen to which this rendering item should be applied. -1 means the bottom row of the host screen.

startCol
The first column of the host screen to which this rendering item should be applied.

startRow
The first row of the host screen to which this rendering item should be applied.

type The host component whose contents will be transformed. The attribute value is the full class name of the host component. There is no default value for this required attribute.

widget
The widget into which the host component will be transformed.

The following tags are also included in each specific rendering item:

componentSettings
The <componentSettings> tag is the enclosing tag for any settings modified for the component for this rendering item. This tag has no attributes.

setting
The <setting> tag is the enclosing tag for any settings modified for the component for this rendering item.

The attributes of the <setting> tag are:

name Specifies the name of a customizable setting for the component. The available settings depend on the component.

Refer to *HATS User's and Administrator's Guide* for descriptions of component settings.

value Specifies the value of a customizable setting for the component. The default values vary depending on the setting.

widgetSettings
The <widgetSettings> tag is the enclosing tag for any settings modified for the widget for this rendering item. This tag has no attributes.

setting
The <setting> tag is the enclosing tag for any settings modified for the widget for this rendering item.

The attributes of the <setting> tag are:

name Specifies the name of a customizable setting for the widget. The available settings depend on the widget.

Refer to *HATS User's and Administrator's Guide* for descriptions of widget settings.

value Specifies the value of a customizable setting for the widget. The default values vary depending on the setting.

textReplacements
The <textReplacements> tag is the enclosing tag for any text replacement specified for this rendering item. This tag has no attributes.

replace
The <replace> tag specifies the text replacement values for this rendering item.

The attributes of the <replace> tag are:

caseSensitive
Specifies whether the case of text replacement values must match before text replacement occurs. Valid values are `true` and `false`. The default is `false`.

from Specifies the text you want to replace. The text on the **from** attribute must be enclosed in quotes.

to Specifies the replacement string you want to insert in place of the value specified on the **from** attribute. The replacement string on the **to** attribute must be enclosed in quotes.

regularExpression
Specifies whether Java regular expression support is used as part of the text replacement algorithm. A regular expression is a pattern of characters that describes a set of strings. You can use regular expressions to find and modify occurrences of a pattern. Valid values are `true` and `false`. The default is `false`.

toImage
When replacing text with an image, specifies the path and name of the image you want to insert in place of the value specified on the **from** attribute. The path and name of the image on the **toImage** attribute must be enclosed in quotes.

matchLTR
When using a bidirectional code page, specifies whether the value specified on the **from** attribute is replaced on left-to-right screens. Valid values are `true` and `false`. The default is `true`.

matchRTL
When using a bidirectional code page, specifies whether the value specified on the **from** attribute is replaced on right-to-left screens. Valid values are `true` and `false`. The default is `false`.

matchReverse
When using a bidirectional code page, specifies whether the value specified on the **from** attribute is replaced when the section of the screen in which it appears has been reversed from the original direction of the page. Valid values are `true` and `false`. The default is `false`.

Note: Care should be taken when using text replacement. Text replacement with a disparate number of characters in the strings can cause changes in the HTML representation of the screen. Depending on the widget used for presenting a region of a screen, text on a line of the screen can be contracted, expanded, or forced to a new line.

<globalRules> tag

The <globalRules> tag is the enclosing tag for any global rules you define in the project. It has no attributes.

<rule> tag

The <rule> tag defines a global rule.

The attributes of the <rule> tag for project-level rules are the same as for screen customization-level global rules. However, when you create a project-level and a

screen customization-level global rule using the same input field, the screen customization-level rule will have a higher priority. The <rule> tag attributes are:

associatedScreen
The name of a screen capture in the project, from which the global rule is defined.

description
The description entered when the global rule was created.

enabled
Indicates whether this global rule is enabled. Reflects the state of the check box on the Rendering page of Project Settings.

endCol
The last column of the host screen to which this global rule should be applied. -1 means the rightmost column of the host screen.

endRow
The last row of the host screen to which this global rule should be applied. -1 means the bottom row of the host screen.

name The name that will be shown in the list of global rules on the Rendering page of Project Settings.

startCol
The first column of the host screen to which this global rule should be applied.

startRow
The first row of the host screen to which this global rule should be applied.

transformationFragment
The name of the transformation fragment file associated with this global rule. This file contains the information specifying how to transform the host component. It will be included in a transformation if the appropriate input fields are present in the host screen.

type The pattern type component for this global rule, taken from the first page of the Create Global Rule wizard. The type can be one of the following:

com.ibm.hats.transform.components. InputFieldByTextPatternComponent
This pattern component recognizes input fields on the host screen based on text near the fields.

com.ibm.hats.transform.components. AllInputFieldsPatternComponent
This pattern component recognizes all input fields on the host screen.

com.ibm.hats.transform.components. InputFieldBySizePatternComponent
This pattern component recognizes input fields on the host screen based on the size of the input fields.

com.ibm.hats.transform.components. InputFieldByPositionPatternComponent
This pattern component recognizes input fields on the host screen by the field's position on the host screen.

The following tags are also included in each specific global rule:

componentSettings

The <componentSettings> tag is the enclosing tag for any settings defined for the pattern type component specified on the type attribute of the <rule> tag. This tag has no attributes.

setting

The <setting> tag is the enclosing tag for any settings defined for the pattern type component specified on the type attribute of the <rule> tag.

The attributes of the <setting> tag are:

name Specifies the name of a customizable setting for the pattern type component. The available settings depend on the component.

- For the com.ibm.hats.transform.components.InputFieldByTextPattern Component, the settings for the name attribute are:

  caseSensitive
  Specifies whether the case of the text on the text setting must match before the pattern is recognized. Valid values are `true` and `false`. The default is `true`.

  immediatelyNextTo
  Specifies which input fields you want to transform.Valid values are:

  true Specifies that only the nearest input field should be transformed.

  false Specifies that all input fields should be transformed.

  The default is `false`.

  location
  Specifies where text in a protected field, as specified on the text setting, must be in relation to input fields for this global rule to be applied. Valid values are:

  ABOVE
  Specifies that the text must be above the input field.

  BELOW
  Specifies that the text must be below the input field.

  LEFT Specifies that the text must be to the left of the input field.

  RIGHT
  Specifies that the text must be to the right of the input field.

  The default is RIGHT.

  text Specifies some text in a protected field of the host screen. Valid values are any text in a protected field on the host screen.

- For the com.ibm.hats.transform.components.AllInputFieldsPattern Component, there are no component settings.

- For the com.ibm.hats.transform.components.InputFieldBySizePattern Component, the setting for the **name** attribute is **fieldSize**.Valid values are the sizes of any input fields on the host screen.
- For the com.ibm.hats.transform.components. InputFieldByPositionPatternComponent, the setting for the **name** attribute is **enableFieldLength**. Valid values are `true` and `false`.

  Note: When **enableFieldLength** is specified, the entire field (as specified by the **fieldSize** attribute) must be within the defined region boundary in order for the field to be recognized. The region boundary is defined by the values for the **startRow**, **endRow**, **startCol** and **endCol** attributes.

Connection files (.hco)

Each connection that you define in a HATS project is represented by a connection file. The connection (.hco) files are stored in the *project_name*`/Connections` folder, where *project_name* is the name you gave the project when you created it. The default connection, which is created using the connection values that you specify in the New HATS Project wizard, is stored in main.hco.

<hodconnection> tag

The <hodconnection> tag begins the connection definition and specifies several characteristics for the connection.

The attributes of the <hodconnection> tag are:

certificateFile
Specifies the name of the file from which the project's SSL certificate was imported, if any.

codePage
Specifies the numeric value for the code page used on this connection. The default value is the value you selected when you created the project. Each connection can use a different code page. See the description of the codePageKey attribute for the code page numbers.

codePageKey
Specifies the usage key that corresponds to the numeric code page. The default value is KEY_US. Valid values for codePage and the location or usage key are:

Table 6. Code pages and usage keys

Code page	Usage key
037	KEY_BELGIUM KEY_BRAZIL KEY_CANADA KEY_NETHERLANDS KEY_PORTUGAL KEY_US
273	KEY_AUSTRIA KEY_GERMANY
274	KEY_BELGIUM_OLD
275	KEY_BRAZIL_OLD

Table 6. Code pages and usage keys (continued)

Code page	Usage key
277	KEY_DENMARK KEY_NORWAY
278	KEY_FINLAND KEY_SWEDEN
280	KEY_ITALY
284	KEY_SPAIN KEY_LATIN_AMERICA
285	KEY_UNITED_KINGDOM
297	KEY_FRANCE
420	KEY_ARABIC
424	KEY_HEBREW
500	KEY_MULTILINGUAL
803	KEY_HEBREW_OLD
838	KEY_THAI
870	KEY_BOSNIA_HERZEGOVINA KEY_CROATIA KEY_CZECH KEY_HUNGARY KEY_POLAND KEY_ROMANIA KEY_SLOVAKIA KEY_SLOVENIA
871	KEY_ICELAND
875	KEY_GREECE
924	KEY_MULTILINGUAL_ISO_EURO
930	KEY_JAPAN_KATAKANA
933	KEY_KOREA_EX
937	KEY_ROC_EX
939	KEY_JAPAN_ENGLISH_EX
1025	KEY_BELARUS KEY_BULGARIA KEY_MACEDONIA KEY_RUSSIA KEY_SERBIA_MONTEGRO
1026	KEY_TURKEY
1047	KEY_OPEN_EDITION
1112	KEY_LATVIA KEY_LITHUANIA
1122	KEY_ESTONIA
1123	KEY_UKRAINE
1137	KEY_HINDI

Table 6. Code pages and usage keys (continued)

Code page	Usage key
1140	KEY_BELGIUM_EURO KEY_BRAZIL_EURO KEY_CANADA_EURO KEY_NETHERLANDS_EURO KEY_PORTUGAL_EURO KEY_US_EURO
1141	KEY_AUSTRIA_EURO KEY_GERMANY_EURO
1142	KEY_DENMARK_EURO KEY_NORWAY_EURO
1143	KEY_FINLAND_EURO KEY_SWEDEN_EURO
1144	KEY_ITALY_EURO
1145	KEY_LATIN_AMERICA_EURO KEY_SPAIN_EURO
1146	KEY_UNITED_KINGDOM_EURO
1147	KEY_FRANCE_EURO
1148	KEY_MULTILINGUAL_EURO
1149	KEY_ICELAND_EURO
1153	KEY_BOSNIA_HERZEGOVINA_EURO KEY_CROATIA_EURO KEY_CZECH_EURO KEY_HUNGARY_EURO KEY_POLAND_EURO KEY_ROMANIA_EURO KEY_SLOVAKIA_EURO KEY_SLOVENIA_EURO
1154	KEY_BELARUS_EURO KEY_BULGARIA_EURO KEY_MACEDONIA_EURO KEY_RUSSIA_EURO KEY_SERBIA_MONTEGRO_EURO
1155	KEY_TURKEY_EURO
1156	KEY_LATVIA_EURO KEY_LITHUANIA_EURO
1157	KEY_ESTONIA_EURO
1158	KEY_UKRAINE_EURO
1160	KEY_THAI_EURO
1166	KEY_KAZAKHSTAN_EURO
1364	KEY_KOREA_EURO
1371	KEY_ROC_EURO
1388	KEY_PRC_EX_GBK
1390	KEY_JAPAN_KATAKANA_EX_EURO
1399	KEY_JAPAN_ENGLISH_EX_EURO

connecttimeout
Specifies the time that HATS attempts to connect to a host. Specify a number of seconds between 1 and 2147483647. The initial default is 120 seconds.

description
Specifies the description for the connection when it was created.

disableFldShp
When using a bidirectional code page, specifies whether you want Arabic data in password fields submitted to the host in isolated form or in shaped form. Valid values are `true` and `false`. There is no initial default.

disableNumSwapSubmit
When using a bidirectional code page, specifies whether you want to disable entry of Arabic-Western numbers, that is, allow entry of only Arabic-Indic numbers in RTL screens. Do this so that, when submitted, all numbers are submitted as Arabic-Western numbers. Valid values are `true` and `false`. There is no initial default.

disconnecttimeout
Specifies the time that HATS attempts to disconnect from a host. Specify a number of seconds in the range of 1-2147483647. The initial default is 120 seconds.

enableScrRev
When using a bidirectional code page, specifies which pages of an application should display a **Screen Reverse** button to enable users to reverse the direction of displayed text and input fields. Valid values are:

(blank)
The **Screen Reverse** button is not placed on any screens.

Customized
The **Screen Reverse** button is placed on screens that match a screen customization and on screens that do not match a screen customization, in other words, on all screens. There is no option to place the **Screen Reverse** button only on screens that match a screen customization.

Non-customized
The **Screen Reverse** button is placed only on screens that do not match a screen customization.

There is no initial default.

host Specifies the name of the host to which the connection is made.

hostSimulationName
Specifies the name of the host simulation trace file to use instead of a live connection.

LUName
Valid only on enhanced 3270 sessions (TNEnhanced="true"). Sets the LUName property, which is the LU name used during enhanced negotiation. Values are in string format. Maximum length of LUName is 17 characters. There is no default. To configure print support for your 3270 HATS project, you must specify that the host type is 3270E. When you add the LUName parameter to the list of connection settings, do not use the printer LU name; use the name of your display LU or a pool of display LUs.

LUNameSource
Valid only on enhanced 3270 sessions (TNEnhanced="true"). Specifies the source of the LU name for the connection. Valid values are:

automatic
The LU name is automatically assigned when the connection is established.

prompt
Prompt the end user for the LU name. If pooling is enabled, prompt should not be used.

session
The LU name is defined using an HTTP session variable. The LUName attribute names the session variable. If pooling is enabled, session should not be used.

value The LU name is defined on the LUName attribute.

There is no initial default.

port Specifies the number of the port through which the connection to the host is made. The valid range for ports is 0–65535. The initial default is 23.

screenSize
Specifies the number of rows and columns that the host terminal displays. Valid values for screenSize are:

- 2=24x80
- 3=32x80
- 4=43x80
- 5=27x132
- 6=24x132 (VT only)

The initial default screen size is 24 x 80.

sessionType
Specifies the type of terminal the host terminal displays. Valid values for type are:

- 1=3270
- 2=5250
- 3=VT

The initial default is 3270.

singlelogon
When user lists are defined in the project, specifies whether a user ID can be used more than once at a time. Valid values are:

true The user ID can be used only once at a time.

false The user ID can connect multiple times simultaneously.

The initial default is `false`.

SSL Specifies whether SSL is enabled. Valid values are:

true SSL is enabled for the project.

false SSL is not enabled for the project.

TNEnhanced
Valid only on 3270 connections. Specifies whether the connection is a TN3270E connection. Valid values are `true` and `false`. The initial default is `true`.

VTTerminalType
Valid only on VT connections. Indicates the type of VT terminal. Valid values are:

- 1=VT420_7
- 2=VT420_8
- 3=VT100
- 4=VT52

WFEnabled
Valid only on 5250 connections. Specifies whether the connection is a 5250W connection. Valid values are `true` and `false`. The initial default is `false`.

workstationID
Valid only on 5250 and 5250W connections. When the workstationIDSource attribute is set to either session or value, specifies the HTTP session variable or the workstation ID for the connection. There is no initial default.

workstationIDSource
Valid only on 5250 and 5250W connections. Specifies the source of the workstation ID for the connection. Valid values are:

automatic
The workstation ID is automatically assigned when the connection is established.

prompt
Prompt the end user for the workstation ID. If pooling is enabled, prompt should not be used.

session
The workstation ID is defined using an HTTP session variable. The workstationID attribute names the session variable. If pooling is enabled, session should not be used.

value The workstation ID is defined on the workstationID attribute.

There is no initial default.

<otherParameters> tag

The <otherParameters> tag specifies additional Host On-Demand session parameters.

Host On-Demand session parameters supported by HATS include:

ENPTUI
Determines whether a project with a connection to a 5250 host can use display data stream (DDS) keywords for the Enhanced Non-Programmable Terminal User Interface (ENPTUI). Valid values are `true` and `false`. The default value is `false`.

HTMLDDS
Determines whether a project with a connection to a 5250 host can use

HTML fragments along with the 5250 display data stream (DDS). This feature is not intended for display with standard emulation programs; it is only sent to 5250 Workstation Gateway devices, and host access products such as HATS, that have been enhanced to show this HTML data in a browser HTML DDS is available as a component when the host type is specified as 5250. The use of this component is not part of the default rendering and must be added to either the default rendering or custom transformations. Valid values are:

- Ignore DDS data using filter
- Accept DDS data using filter

For further information, see the HATS User's and Administrator's Guide or HATS Knowledge Center at http://www.ibm.com/support/knowledgecenter/SSXKAY_9.5.0.

Lamalef
Sets the LamAlef property, which determines whether LamAlef should be expanded or compressed. This property applies to Arabic sessions only. Values are in string format. Valid values are:

- LAMALEF_ON
- LAMALEF_OFF

The default value is LAMALEF_OFF.

numeralShape
Sets the numeralShape property. This property applies to bidirectional sessions only. Values are in string format. The default value is NOMINAL.

numericSwapEnabled
Sets the Numeric swapping property. This property applies to Arabic 3270 sessions only. Valid values are `true` and `false`. The default value is `true`.

roundTrip
Sets the roundTrip property. This property applies to bidirectional sessions only. Values are in string format. Valid values are:

- ROUNDTRIP_ON
- ROUNDTRIP_OFF

The default value is ROUNDTRIP_ON.

SecurityProtocol
Sets the SecurityProtocol property, which indicates whether to use the TLS v1.0 protocol or the SSL protocol for providing security. Values are in string format. The default value is TLS.

SSLServerAuthentication
Sets the SSLServerAuthentication property, which indicates whether SSL server authentication is enabled. Valid values are `true` and `false`. The default value is `false`.

symmetricSwapEnabled
Sets the symmetric swapping property. This property applies to Arabic 3270 sessions only. Valid values are `true` and `false`. The default value is `true`.

textOrientation
Sets the textOrientation property. This property applies to bidirectional sessions only. Values are in string format. Valid values are:

- LEFT_TO_RIGHT

- RIGHT_TO_LEFT

The default value is LEFT_TO_RIGHT.

ThaiDisplayMode
Sets the Thai display mode property. This property applies to Thai sessions only. Values are in string format. The default value is THAI_MODE_5.

workstationID
Sets the workstationID property, which is used during enhanced negotiation for 5250. Values are in string format. All lowercase characters are converted to uppercase. There is no default value.

<classSettings> tag

The <classSettings> tag is the enclosing tag for the Java classes you include in the connection definition.

<class> tag

The <class> tag specifies a class whose attributes are defined in the enclosed <settings> tags.

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

name Specifies one of the following Java classes:

- com.ibm.hats.common.HATSPrintSettings
- com.ibm.hats.common.NextScreenSettings

The class names on the name attribute must be enclosed in quotes.

<setting> tag

The <setting> tag specifies the settings associated with the class in which the <setting> tag is enclosed.

The attributes of the <setting> tag are:

name Specifies the name of a customizable setting for the class defined by the name attribute of the <class> tag. The available settings depend on the class.

For the com.ibm.hats.common.HATSPrintSettings class, the customizable settings are:

printFontName
Specifies the font in which you want your output printed. Valid values depend on the value of the codePage attribute.

printNumSwapSupport
Specifies whether numeric swapping is enabled. This property applies to Arabic 3270 sessions only, when printRTLSupport is enabled. English numerals are replaced by Arabic numerals in right-to-left screens and Arabic numerals are replaced by English numerals in right-to-left Screens. Valid values are `true` and `false`. The default value is `true`.

printOrientation
Specifies how your printed output is positioned on the page. Valid values for printOrientation are:

PDF_ORIENTATION_PORTRAIT
Orients the paper vertically.

PDF_ORIENTATION_LANDSCAPE
Rotates the paper 90 degrees clockwise.

printPaperSize
Specifies the size of the paper on which to print your output. Valid values for printPaperSize are:

ISO_A3
ISO/DN & JIS A4, 297 x 420 mm

ISO_A4
ISO/DN & JIS A4, 210 x 297 mm

ISO_A5
ISO/DN & JIS A4, 148 x 210 mm

ISO_B4
ISO/DN B4, 250 x 353 mm

ISO_B5
ISO/DN B5, 176 x 250 mm

JIS_B4
JIS B4, 257 x 364 mm

JIS_B5
JIS B5, 182 x 257 mm

ISO_C5
ISO/DN C5, 162 x 229 mm

ISO_DESIGNATED_LONG
ISO/DN Designated Long, 110 x 220 mm

EXECUTIVE
Executive, 7 1/4 x 10 1/2 in

LEDGER
Ledger, 11 x 17 in

NA_LETTER
North American Letter, 8 1/2 x 11 in

NA_LEGAL
North American Legal, 8 1/2 x 14 in

NA_NUMBER_9_ENVELOPE
North American #9 Business Envelope, 3 7/8 x 8 7/8 in

NA_NUMBER_10_ENVELOPE
North American #10 Business Envelope, 4 1/8 x 9 1/2 in

MONARCH_ENVELOPE
Monarch Envelope, 3 7/8 x 7 1/2 in

CONTINUOUS_80_COLUMNS
Data Processing 80 Columns Continuous Sheet, 8 x 11 in

CONTINUOUS_132_COLUMNS
Data Processing 132 Columns Continuous Sheet, 13 1/5 x 11 in

printRTLSupport
Specifies whether right-to-left print support is enabled. This property applies to Arabic 3270 sessions only. Bidirectional files can be either RTL or LTR files. Valid values are `true` and `false`. The default value is `true`.

printSupport
Specifies whether your project includes print capability. Valid values for printSupport are `true` and `false`. The initial default is `false`.

printSymSwapSupport
Specifies whether symmetric swapping is enabled; swapping characters are swapped in right-to-left screens. This property applies to Arabic 3270 sessions only, when printRTLSupport is enabled. Valid values are `true` and `false`. The default value is `true`.

printURL
Specifies the URL for a System i® Access for Web Printer Output window on a 5250 server. The default URL is http://*hostname*/webaccess/iWASpool, where *hostname* is the name of the 5250 server.

The customizable settings for the com.ibm.hats.common.NextScreenSettings class are:

default.appletDelayInterval
Specifies the maximum time (in milliseconds) that the server waits until a full host screen has arrived for a session running in asynchronous update mode. The initial default value is 400 milliseconds.

default.blankScreen
Specifies how to handle a blank screen received at connection startup. Valid values are:

normal
Display the blank screen.

sendkeys
Send the host key defined on the default.blankScreen.keys setting.

timeout
Wait for the connection to time out before issuing an error message.

The default is `normal`.

default.blankScreen.keys
Specifies the host key to send when default.blankScreen is set to sendkeys.

default.delayInterval
Specifies the maximum time, in milliseconds, that the server waits for the arrival of screen updates after the initial screen update. The initial default value is 1200 milliseconds.

default.delayStart
Specifies the minimum time (in milliseconds) that the server waits

until the first full host screen has arrived after the host connection becomes ready. The initial default value is 2000 milliseconds.

nextScreenClass
Specifies a class that turns off the default, speed-optimized, algorithm in favor of accuracy. The class for the value attribute is com.ibm.hats.runtime.TimingNextScreenBean. As a result, screen transitions might be slower. The setting `default.delayInterval` is now the minimum amount of time (in milliseconds) per screen transition. The `default.delayInterval` value has a default of 1200 milliseconds, but you can customize it for your network and your host application. If you raise this value, remember that HATS waits at least this long for the host screen to settle.

oiaLockMaxWait
Specifies the maximum time (in milliseconds) that HATS should wait after the host screen has settled to ensure that the OIA system lock status has been released. The value can be in the range of 0–600000 milliseconds. The initial default value is 300000 milliseconds.

value The values for the settings are included in the description of the individual settings.

<poolsettings> tag

The <poolsettings> tag defines pooling parameters for the connection.

The attributes of the <poolsettings> tag are:

enabled
Specifies whether pooling is enabled for this connection. Valid values for enabled are `true` and `false`. The initial default is `false`.

maxbusytime
The number of seconds before a connection that is in use by an end user will be terminated. If you do not want active connections to end, set this field to -1. This setting is available for connections that have pooling enabled as well as for connections that have pooling disabled. For a connection with pooling enabled, the connection returns to the pool if the number of available connections in the pool is less than the minimum number of connections you specified to remain connected. Otherwise, this connection is discarded. For a connection with pooling disabled, the connection is discarded. Valid number of seconds is -1 or in the range of 60-2147483647. The default is -1 (no maximum busy time).

maxconnections
The maximum number of connections in the pool that can be active. This setting is available only for connections that have pooling enabled. Valid number of connections is in the range of 1-2147483647. The default is 1. When you reach the maximum specified and an additional request for a connection is received, HATS can either wait for the next available connection or create a new connection.

maxidletime
The number of seconds before a connection that is not in use by an end user will be terminated and removed from the pool. If you do not want inactive connections to end, set this field to -1. This setting is available only for connections with connection pooling enabled. The minimum number of connections you specify remain connected, whether or not they

are used. Valid number of seconds is -1 or in the range of 60-2147483647. The default is -1 (no maximum idle time).

minconnections
The number of idle connections in the pool that remain connected. This setting is available only for connections that have pooling enabled and have the maxidletime before disconnection set to some value other than -1. Valid number of connections is between 0 and 2147483647. The default is 0 (do not keep connections connected).

overflowallowed
Whether a new, non-pooled connection should be created if the maximum limit of connections has been reached. If this value is false, you must specify the number of seconds to wait for a pooled connection to become available. If the time to wait elapses and a connection does not become available, HATS returns an error. If this value is true, a new, non-pooled connection will be created. When the end user finishes with this type of connection, it is not put back in the pool, but discarded.

waittimeout
The number of seconds to wait for a pooled connection to become available once the maximum limit of connections has been reached, and another request comes in. Valid number of seconds is between 0 and 2147483647, or -1 if you want to wait forever. The default is 120.

<webexpresslogon> tag

The <webexpresslogon> tag indicates whether the Web Express Logon function is enabled for this connection.

The attribute of the <webexpresslogon> tag is:

enabled
Specifies whether the Web Express Logon function is enabled for this connection. Valid values for enabled are `true` and `false`. The initial default is `false`.

<userconfig> tag

The <userconfig> tag defines a user list for the connection. The tags and data within the <userconfig> tag are complex and can be corrupted by manual editing. To protect the integrity of your user list, do not manually edit the <userconfig> data. Instead, use the **User List** tab on the Connection editor to create or modify a user list. By default, a host connection does not specify this tag and does not have a user list.

Template and transformation files (.jsp)

These JavaServer Pages (JSP) files contain HTML and JSP tagging to define how your project appears in the end user's browser.

The template .jsp files are stored in the *project_name*/**Web Content/Templates** directory. The transformation .jsp files are stored in the *project_name*/**Web Content/Transformations** directory. You can view and edit the source of the .jsp files by double-clicking on the name of the template or transformation in the **HATS Project View** to open the JSP editor. The source for the file can be viewed by clicking on the **Source** tab.

You can modify the template and transformation files using either the **Design** or the **Source** tabs in the JSP editor. When you make a change on one tab, the affected information on the other tab is automatically updated.

A template .jsp file contains HTML tagging to define links and images for the project page. The template .jsp file also contains a <HATS:Transform> tag that defines the transformation to be used with the template to present the page of your project.

A transformation .jsp file contains HTML tags to describe the layout of the information presented to the user of the project in a Web browser. The transformation .jsp file may also contain <HATS:Component> tags that define HATS components and widgets used to present the page of your project. For more information on the <HATS:Component> tag, see "HATS component tag and attributes" on page 17.

Screen combination files (.evnt)

The screen combination files defines how a host screen is recognized, the actions HATS performs when a screen is recognized, how to define the end of the screen combination, and how to navigate between screens. The screen combination (.evnt) files are stored in the *project_name*/**Web Content/WEB-INF/profiles/events/screencombinations** directory. You can view and edit the source of the screen combination files by double-clicking on the name of the screen combination in the HATS Projects view to open the screen combination editor. The source for the file can be viewed by clicking on the **Source** tab. You can modify screen combination files using the **Begin Screen**, **Render**, **Navigation**, **End Screen**, **Actions**, **Text replacement**, or **Source** tabs in the editor. HATS Toolkit updates the affected information on other tabs when you make changes on any tab. The screen combination event files contain tags to define how a host screen is recognized and the actions and navigations that will occur when the host screen is recognized.

Screen combination adds several tags to those found in screen customization (.evnt) files.

<combinations> tag

This is the container for the combination information. It consists of a type attribute and a rendering item detailing the screen combination component.

type The type parameter determines how the screen transformation will be aggregated.

If the string value is set to *dynamic*, the screen transformation can add screens to the combined area while the user is using the screen transformation.

If the string value is set to *normal* or is missing, the individual screens compound prior to allowing the user to interact with the screen transformation. Rich Client screen combinations are limited to normal processing.

<enddescription> tag

This is the description for the screen criteria used to determine if the screen combination end screen has been reached. The tags and details match the description tag. It has an attribute associatedScreen for the screen associated with the end screen.

associatedScreen
This is the screen capture associated with the end screen.

<navigation> tag

The navigation contains the commands needed to move between screens to gather and place data. It consists of a screenUp and screenDown tag.

<screenUp> tag

The commands necessary to traverse to a screen backward in the combination. This is used to return data to the correct place in a screen combination. It can consist of keyPress, setCursor, and sendText tags.

<screenDown> tag

The commands necessary to traverse to a screen forward in the combination. This is used to create the screen combination view as well as return data to the correct place in a screen combination. It can consist of keyPress, setCursor, and sendText tags.

<keyPress> tag

This navigation command is the equivalent of a sendKey. It has a value attribute, which must be a valid HOD key command, for the HOD command to send.

value The value attribute for the keyPress tag should be a valid HOD key command.

<setCursor> tag

This navigation command allows cursor positioning on the screen. It has a row and a column attribute for the cursor positioning.

row The row attribute should be a 1-based integer that equates to a position on the screen. This positions the vertical component of the cursor position.

column
The column attribute should be a 1-based integer that equates to a position on the screen. This positions the horizontal component of the cursor position.

<sendText>

This navigation command is the equivalent of a sendKey. It has a value attribute, which must be valid text for the host field, for the text to send.

value The value attribute for the sendText tag should be valid text for the host field.

Screen customization files (.evnt)

The screen customization files define how a host screen is recognized, and also defines the actions HATS performs when a screen is recognized.

The screen customization (.evnt) files are stored in the *project_name*/**Web Content/WEB-INF/profiles/events/screencustomizations** directory. You can view and edit the source of the screen customization files by double-clicking on the name of the screen customization in the **HATS Project View** to open the screen customization editor. The source for the file can be viewed by clicking on the **Source** tab.

You can modify screen customization files using the **Screen Recognition Criteria**, **Actions**, **Text replacement**, or **Source** tabs in the editor. HATS Toolkit updates the affected information on other tabs when you make changes on any tab.

The screen customization event files contain tags to define how a host screen is recognized and the actions that will occur when the host screen is recognized.

<event> tag

Begins the definition of the screen customization. The event tag has the following attributes:

description
If you supplied a description of the screen customization when you created it, that description is defined in this attribute.

type For a screen customization, type is always screenRecognize. For combined screens, type is screenCombination.

<actions> tag

This is the enclosing tag for all of the actions defined for a screen customization. It has no attributes.

<apply> tag

Defines the action for applying a transformation. The attributes of the <apply> tag are:

applyGlobalRules
Specifies whether HATS should look for global rules on this host screen. Default is `true`.

applyTextReplacement
Specifies whether HATS should look for text replacements on this host screen. Default is `true`. See the <replace> tag for further information on how to use text replacement.

enabled
Indicates whether this action is enabled for use. The default is `true`.

immediateKeyset
Defines the host keys sent to the host immediately when pressed by the end user of your project. If you did not define any host keys to be sent to the host immediately, this attribute has an empty value.

template
Names the template file that surrounds the transformation being applied. If the default template is being used to surround the transformation, this attribute has an empty value.

transformation
Names the transformation file that is to be applied for this action.

<insert> tag

Defines the action for inserting a global variable or a string. The attributes of the <insert> tag are:

enabled
Indicates whether this action is enabled for use. The default is `true`.

row Defines the starting row on the host screen where the value is to be inserted.

col Defines the starting column on the host screen where the value is to be inserted.

source Specifies whether the value to be inserted is a string or the value of a global variable. Valid values are string and variable.

value Specifies either the string to be inserted onto the host screen or the name of a global variable from which the value is taken.

fill If the source of the value to be inserted is an indexed global variable, fill specifies whether the indices of the global variable are to be concatenated and inserted at the specified position, or inserted into a rectangular region of the host screen. Valid values are concatenate and rectangular.

index If the source of the value to be inserted is an indexed global variable, index specifies the number of the index that is to be used as the value to be inserted onto the host screen.

shared
If the source of the value to be inserted is a global variable, shared specifies whether this global variable is shared between all the applications in the same EAR file.

<extract> tag

Defines the action for extracting a global variable. The attributes of the <extract> tag are:

enabled
Indicates whether this action is enabled for use. The default is `true`.

srow Defines the starting row on the host screen of the text being extracted.

erow Defines the ending row on the host screen of the text being extracted.

scol Defines the starting column on the host screen of the text being extracted.

ecol Defines the ending column on the host screen of the text being extracted.

name Specifies the name of the global variable to which the text is extracted. This can be an existing global variable or a new global variable.

overwrite
Specifies whether the text extracted is to overwrite the value of an existing global variable. Valid values are `true` and `false`.

indexed
Specifies whether the text extracted is a single string or a list of strings, where each string in the list corresponds to a single row of text from the extracted region. Valid values are true and false.

index If an existing global variable is indexed, this attribute specifies the index number to which the extracted value is to be written. The effect of this attribute is dependent on the value of the **overwrite** attribute. If `overwrite=true`, the extracted value overwrites the existing variable, starting at the specified index. If `overwrite=false`, the extracted value is inserted into the existing variable, beginning at the specified index.

shared
Shared specifies whether the global variable is shared between all the applications in the same EAR file.

<set> tag

Defines the action for setting a global variable. The attributes of the <set> tag are:

enabled
Indicates whether this action is enabled for use. The default is true.

name Specifies the name of the global variable being set. This can be an existing global variable or a new global variable.

shared
Specifies whether the global variable being set is shared between all the applications in the same EAR file.

type Specifies whether the value of the global variable being set comes from a fixed constant or a calculated value. Valid values are `string` and `calculate`.

value Specifies the value being assigned to the global variable.

overwrite
Specifies whether the value being set is to overwrite the value of an existing global variable. Valid values are `true` and `false`.

index If the value being set is being written to an existing indexed global variable, this attribute specifies the index number to which the value being set is written. The effect of this attribute is dependent on the value of the **overwrite** attribute. If `overwrite=true`, the value being set overwrites the existing variable, beginning at the specified index. If `overwrite=false`, the value being set is inserted into the existing variable, beginning at the specified index.

op1 Specifies whether the first operand of a calculated value is a fixed constant or the value of an existing global variable. Valid values are a fixed constant or the name of a global variable.

op1_type
Specifies whether the value of the first operand of a calculated value is set as a fixed constant or from an existing global variable. Valid values are `string` and `variable`.

op1_index
If the source of the value of the first operand of a calculated value is an indexed global variable, op1_index specifies the number of the index used as the value for the calculation.

op1_shared
If the value of op1 is a global variable, shared specifies whether this global variable is shared between all the applications in the same EAR file.

op Specifies the type of operation to occur between the first and second operands of a calculated value. Valid values are concatenate, + (add), - (subtract), * (multiply), / (divide), and % (modulo).

op2 Specifies whether the second operand of a calculated value is a fixed constant or the value of an existing global variable. Valid values are a fixed constant or the name of a global variable.

op2_type
Specifies whether the value of the second operand of a calculated value is set as a fixed constant or from an existing global variable. Valid values are `string` and `variable`.

op2_index
If the source of the value of the second operand of a calculated value is an indexed global variable, op2_index specifies the number of the index used as the value for the calculation.

op2_shared
If the value of op2 is a global variable, shared specifies whether this global variable is shared between all the applications in the same EAR file.

dec Specifies the number of decimal places to which a calculated value is rounded. Valid values are 0–999.

<execute> tag

Defines the action for executing business logic. The attributes of the <execute> tag are:

enabled
Indicates whether this action is enabled for use. The default is `true`.

class Names the Java class that contains your business logic. The class value is required.

method
Names the method inside the class that executes the business logic. The method value is required.

package
Names the package that the Java class resides in on your file system. The package value is optional.

<show> tag

Defines the action for showing a URL. The <show> tag has the following attributes:

enabled
Indicates whether this action is enabled for use. The default is `true`.

template
Specifies the template to be used for this action.

url Identifies the Uniform Resource Locator (URL) of the Web page to show. This attribute is required.

<forwardtoURL> tag

Defines the action for passing control from a project to a JSP that invokes an Integration Object. The <forwardtoURL> tag has the following attributes:

enabled
Indicates whether this action is enabled for use. The default is `true`.

startStateLabel
If forwarding control to a JSP with an Integration Object chain, specifies the start state label of the first Integration Object in the chain to be executed.

url Specifies the URL of the Integration Object JSP.

<disconnect> tag

Disconnects the default connection. Use this action carefully and only for events from which you cannot recover. The <disconnect> tag has the following attribute:

enabled
Indicates whether this action is enabled for use. The default is `true`.

<play> tag

Defines the action for playing a macro. The <play> tag has the following attributes:

enabled
Indicates whether this action is enabled for use. The default is `true`.

macro Names the macro to be played. This attribute is required.

<perform> tag

Defines the action for playing a macro on any connection, not necessarily the default connection. This action does not affect the current host screen. The <perform> tag has the following attributes:

connection
The connection on which the macro is to be played. The default is `main`.

enabled
Indicates whether this action is enabled for use. The default is `true`.

macro The name of the macro to be played.

<pause> tag

Defines the action for waiting for some time before continuing with normal processing. The <pause> tag has the following attributes:

enabled
Indicates whether this action is enabled for use. The default is `true`.

time Specifies the time, in milliseconds, to pause before continuing with normal processing.

<sendkey> tag

Defines the action for sending a specified key to the host screen to perform an action. The <sendkey> tag has the following attributes:

enabled
Indicates whether this action is enabled for use. The default is `true`.

key Indicates key to send to the host screen.

row Defines the starting row on the host screen where the key is to be inserted.

col Defines the starting column on the host screen where the key is to be inserted.

<globalRules> tag

The <globalRules> tag is the enclosing tag for any global rules you define for screen events. It has no attributes.

<rule> tag

The <rule> tag defines a global rule.

The attributes of the <rule> tag for screen customization-level rules are the same as for project-level global rules. However, when you create a screen

customization-level and a project-level global rule using the same input field, the screen customization-level rule will have a higher priority. The <rule> tag attributes are:

associatedScreen
The name of a screen capture in the project, from which the global rule is defined.

componentSettings
Any settings configured for the global rule, such as recognition criteria.

description
The description entered when the global rule was created.

enabled
Indicates whether this global rule is enabled. Reflects the state of the check box on the Rendering page of Project Settings.

endCol
The last column of the host screen to which this global rule should be applied. -1 means the rightmost column of the host screen.

endRow
The last row of the host screen to which this global rule should be applied. -1 means the bottom row of the host screen.

name The name that will be shown in the list of global rules on the Rendering page of Project Settings.

startCol
The first column of the host screen to which this global rule should be applied.

startRow
The first row of the host screen to which this global rule should be applied.

transformationFragment
The name of the transformation fragment file associated with this global rule. This file contains the information specifying how to transform the host component. It will be included in a transformation if the appropriate input fields are present in the host screen.

type The pattern type component for this global rule, taken from the first page of the Create Global Rule wizard. The type can be one of the following:

com.ibm.hats.transform.components.InputFieldByTextPatternComponent
This pattern component recognizes input fields on the host screen based on text near the fields.

com.ibm.hats.transform.components.AllInputFieldsPatternComponent
This pattern component recognizes all input fields on the host screen.

com.ibm.hats.transform.components.InputFieldBySizePatternComponent
This pattern component recognizes input fields on the host screen based on the size of the input fields.

The following tags are also included in each specific global rule:

componentSettings

The <componentSettings> tag is the enclosing tag for any settings defined for the pattern type component specified on the type attribute of the <rule> tag. This tag has no attributes.

setting

The <setting> tag is the enclosing tag for any settings defined for the pattern type component specified on the type attribute of the <rule> tag.

The attributes of the <setting> tag are:

name Specifies the name of a customizable setting for the pattern type component. The available settings depend on the component.

- For the com.ibm.hats.transform.components.InputFieldByTextPattern Component, the settings for the name attribute are:

  caseSensitive
  Specifies whether the case of the text on the text setting must match before the pattern is recognized. Valid values are `true` and `false`. The default is `true`.

  immediatelyNextTo
  Specifies which input fields you want to transform.Valid values are:

  true Specifies that only the nearest input field should be transformed.

  false Specifies that all input fields should be transformed.

  The default is `false`.

  location
  Specifies where text in a protected field, as specified on the text setting, must be in relation to input fields for this global rule to be applied. Valid values are:

  ABOVE
  Specifies that the text must be above the input field.

  BELOW
  Specifies that the text must be below the input field.

  LEFT Specifies that the text must be to the left of the input field.

  RIGHT
  Specifies that the text must be to the right of the input field.

  The default is RIGHT.

  text Specifies some text in a protected field of the host screen. Valid values are any text in a protected field on the host screen.

- For the com.ibm.hats.transform.components.AllInputFieldsPattern Component, there are no component settings.

- For the com.ibm.hats.transform.components. InputFieldBySizePattern Component, the setting for the name attribute is fieldSize.Valid values are the sizes of any input fields on the host screen.

<associatedScreens> tag

The <associatedScreens> tag encompasses the screen tag that follows.

<screen> tag

Defines a screen associated with the screen customization. The <screen> tag has the following attribute:

name Specifies the name of a captured screen, for which the screen recognition criteria and actions have been defined.

<description> tag

The <description> tag is the enclosing tag for the description associated with the screen customization, which consists of the <oia> tag and the <string> tag. There are no attributes for the description tag.

<oia> tag

The <oia> tag in the screen customization .evnt file specifies an operator information area (OIA) condition to match. This tag is optional. The default is to wait for inhibit status. The attributes of the <oia> tag are:

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

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

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

<string> tag

The <string> tag describes the screen based on a string. The attributes of the <string> tag are:

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

row The starting row position for a string at an absolute position or in a rectangle. The value must be a number or an expression that evaluates to 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 or an expression that evaluates to a number. This attribute is optional.

erow The ending row position for string in a rectangle. The value must be a number or an expression that evaluates to a number. This attribute is optional. If both **erow** and **ecol** are specified, string is in a rectangle.

ecol The ending column position for string in a rectangle. The value must be a number or an expression that evaluates to a number. This attribute is optional. If both **erow** and **ecol** are specified, string is in a rectangle.

casesense
If `true`, string comparison is case sensitive. The value must be `true` or `false` or an expression that evaluates to true or false. This attribute is optional. The default is `false`.

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

<nextEvents> tag

The <nextEvents> tag encompasses the <event> tag that follows. The <nextEvents> tag has the following attribute:

defaultEvent
Specifies the default screen customization (event) used as the next screen to occur, if there are no matching screen customizations named on the event tags. If defaultEvent does not specify an event, the normal event priority list in the project settings is used. Valid values are:

- `unmatchedScreen`
- `error`
- `disconnect`
- `stop`
- (no value)

<event> tag

Defines another screen customization in the project that is the probable next screen to occur. The <event> tag has the following attributes:

enabled
Indicates whether the screen customization (event) defined on the name attribute is enabled for use. The default is true.

name Specifies the name of a screen customization that is the probable next screen to occur.

<remove> tag

The <remove> tag removes global variables previously added to the screen customization (event). The <remove> tag has the following attributes:

enabled
Indicates whether the global variable defined on the name attribute is enabled for removal. The default is true.

name Specifies the name of the global variable to be removed.

remove Type
Specifies the type of the global variable to be removed. Types include oneLocal, oneShared, allLocal, allShared, and all.

Macro files (.hma)

Macro files are stored in the *project_name*/**Web Content/WEB-INF/profiles/macros** directory. You can view and edit the source of the macro files by double-clicking on the name of the macro in the **HATS Project View** to open the macro editor. The source for the file can be viewed by clicking on the **Source** tab. For more information about macros, see Advanced Macro Guide.

Macro files contain tags that define a set of screens. The tags are described in the sections that follow.

<macro> tag

Begins the definition of the macro. The macro tag has no attributes.

<associatedConnections> tag

The <associatedConnections> tag encompasses the <connection> tag that follows. The attribute of the <associatedConnections> tag is:

default
Identifies the default connection for this macro.

<connection> tag

The <connection> tag identifies the connection with which this macro is associated. The attribute of the connection tag is:

name Identifies the name of the connection with which this macro is associated.

<extracts> tag

The <extracts> tag encompasses the extract tag that follows. The <extracts> tag has no attributes.

<extract> tag

The <extract> tag defines the extraction to occur. The attributes of the extract tag are:

name Specifies the name of the extraction.

handler
You can select a .jsp file to display the extracted information to the end user. A default macro handler is shipped with HATS, and it is named default.jsp. You can find this file by clicking the **HATS Project View** tab of the HATS Toolkit and expanding the project name, and then expanding **Macros > Macro Event Handlers**. If you want to create your own handler, ensure that you return control to the HATS runtime.

showHandler
Specifies whether the extracted information should be shown to the end user. Valid values are true and false.

shared
Specifies whether a global variable being extracted is shared between all the applications in the same EAR file.

save Specifies whether the extracted information is saved to a global variable. Valid values are `true` and `false`.

variableName
If the extracted information is being saved to a global variable, variableName specifies the name of a new or existing global variable.

overwrite
If the extracted information is being saved to an existing global variable, overwrite specifies whether the extracted information is to overwrite the current value of the existing global variable, or whether the extracted information is to be appended to the current value. Valid values are `true` and `false`. `True` specifies that the value of the existing global variable is overwritten.

index If the value being extracted is being written to an existing indexed global variable, this attribute specifies the index number to which the value being set is written. The effect of this attribute is dependent on the value of the **overwrite** attribute. If overwrite=true, the value being extracted overwrites the existing variable, beginning at the specified index. If `overwrite=false`, the value being extracted is inserted into the existing variable, beginning at the specified index.

indexed
Specifies whether the extracted information is a single string or a list of strings. Valid values are `true` and `false`. `True` specifies that the extracted information is a list of strings.

isBidi Specifies whether the connection used in recording the macro is bidirectional. Valid values are `true` and `false`.

isRtlScreen
Specifies whether the bidirectional screen is right-to-left. Valid values are `true` and `false`.

screenorientation
Specifies the orientation of the extract action. Valid values are `ltr` and `rtl`.

<prompts> tag

The <prompts> tag encompasses the prompt tag that follows. The prompts tag has no attributes.

<prompt> tag

The <prompt> tag defines the prompt to occur. The attributes of the <prompt> tag are:

name Specifies the name of the prompt.

handler
You can select a .jsp file to prompt the end user for the necessary information, and include a button for the user to submit the information. A default macro handler is shipped with HATS, and it is named default.jsp. You can find this file by clicking the **HATS Project View** tab of the HATS Toolkit and expanding the project name, and expanding **Macros > Macro Event Handlers**. If you want to create your own handler, ensure that you return control to the HATS runtime.

source Specifies whether the value of the prompt is set to a string or the value of a global variable. Valid values are `string` and `variable`.

variableName
If the value of the prompt is being saved to a global variable, variableName specifies the name of a new or existing global variable.

variableIndex
If the value of the prompt is being saved to an indexed global variable, variableIndex specifies to which index the value should be assigned. This value is always `0`.

variableIndexed
Specifies whether the information for the prompt is coming from an indexed global variable. Valid values are `true` and `false`. `True` specifies that the global variable is indexed.

value Specifies either the string to be used for the prompt or the name of a global variable from which the value is taken.

welApplID
Specifies the application ID to use with the WEL logon macro.

welIsPassword
Specifies whether this is a password field to use with the WEL logon macro.

LTRImpicitOrient
Specifies whether the implicit bidirectional screen orientation is left-to-right. Valid values are `true` and `false`.

isBidi Specifies whether the connection used in recording the macro is bidirectional. Valid values are `true` and `false`.

isRtlField
Specifies whether the bidirectional field is right-to-left. Valid values are `true` and `false`.

isRtlScreen
Specifies whether the bidirectional screen is right-to-left. Valid values are `true` and `false`.

screenorientation
Specifies the orientation of the prompt action. Valid values are `ltr` and `rtl`.

<HAScript> tag

The <HAScript> tag is the main enclosing tag for the other macro tags and attributes. See the *HATS Advanced Macro Guide* for more information about macro tags.

Screen capture files (.hsc)

Screen capture files are XML representations of host screens, used to create or customize screen customizations, screen combinations, transformations, global rules, or macros.

Screen capture files are stored in the *project_name*/Screen Captures directory. You can view these files by double-clicking on the name of the screen capture in the **HATS Project View**. You cannot edit screen capture files.

Note: Screen captures of video terminal (VT) host screens can be used to create or customize a macro using the Visual Macro Editor and as the check-in screen when configuring pooling. They cannot be used to create screen customizations, screen combinations, transformations, default rendering, or global rules.

BMS Map files (.bms and .bmc)

Basic Mapping Support (BMS) maps are screen definitions files for Customer Information Control System (CICS®). Each map defines all or part of a screen, and a CICS application typically displays one or more maps to create a complete screen image. The source for BMS maps is organized in groups called map sets. One map set contains one or more maps. Map sets exist in source form as one map set per source file.

BMS map set files can be imported into a project in HATS Toolkit. When HATS imports BMS maps, the import takes place at the map set level. It is not possible to import an individual map. Imported BMS map set files have a file extension of .bms, and the individual maps have a file extension of .bmc in HATS Toolkit.

Both the map set files (.bms) and the map files (.bmc) are stored in a separate **Maps** folder within the HATS project. By default, the **Maps** folder is not visible in the **HATS Project View** until there are maps imported.

HATS enables you to generate screen captures from map files. You can choose to generate the screen captures when you import map sets, or you can generate them from the maps after they are in the **Maps** folder. To generate screen captures from maps, right click on a map file to display the pop-up menu and select **Generate Screen Captures**. You can elect to create separate screen captures for each BMS map selected or merge selected BMS maps into a single screen capture. Maps cannot be merged if fields overlap. Once the screen captures are created, you can begin creating HATS screen customizations.

You can open a map set file in a HATS Toolkit editor by double-clicking on the file in the **HATS Project View**. See the *CICS Application Programming Reference* for information about the contents of the file. When a map set file is modified and saved in the text editor, the maps that make up the file are regenerated, with one

exception: map files in which the contents of fields defined with labels in the map set files have been modified. To regenerate those maps, you must import the source file again using the BMS map set import wizard.

When a CICS application runs, it can modify the contents of the fields defined with labels. You might need to create screen captures with the fields appearing as they will be when the CICS application runs. Since the labeled fields are changeable when the application runs, the map set file (and the map files that are in the map set) may not contain all the information needed to generate an actual screen capture. While you cannot edit map files, double-clicking on a file opens a file preview. The property sheet view in HATS Toolkit enables you to add missing information and manually set the contents of the fields. By modifying the contents of the fields, a single map can be used for multiple screen captures.

Notes:

1. When you are previewing a map file in HATS Toolkit, the fields displayed in the property sheet view are the fields for the map file highlighted in the **HATS Project View**, not the map file you see in the preview window.
2. You can also screen capture files using the property sheet view in HATS Toolkit, as long as the screen capture files were generated from BMS map files.

Image files (.gif, .jpg, or .png)

Image files are used in HATS Toolkit within template files to create the Web page displayed to the user of your project.

Image files are stored in the *project_name*/**Web Content/common/images** directory. You can view the image files by double-clicking on the name of the image.

Stylesheet files (.css)

Cascading Style Sheets (or style sheets) are used in HATS Toolkit within template files to specify appearance items such as colors, fonts, borders, whitespace, images, margins, and spacing between lines.

The stylesheet files provided by HATS can be grouped into categories.

Unique templates

Some HATS templates use individualized style sheets. The following style sheets are used by the Finance, Industry, Medical, Research, and Transport templates, respectively.

- finance.css
- industry.css
- medical.css
- research.css
- transport.css

In addition, you can use these style sheets in combination with the “Font” and “Unique” style sheets listed below. For example, you can use a font style sheet to override the default defined fonts. You should not use the “Main” or “Reverse video” style sheets listed below in combination with these unique template style sheets.

Main Some HATS templates use a combination of general purpose style sheets. These general purpose main style sheets determine the overall appearance of the template and other controls that you might add to a project.

Examples of these controls include buttons, input fields, tables, fields, and links. These style sheets are named with `theme` at the end, such as blacktheme, whitetheme, monochrometheme, and tantheme. The appearance of the controls in a project is determined by classes named in the style sheets, for example, HATSCHECKBOX, HATSRADIOBUTTON, and HATSDROPDOWN.

Reverse video
Templates that use a general purpose main style sheet, also use a general purpose secondary style sheet that determines the color scheme of any reverse video items in a project. These reverse video style sheets are named with `reverseVideo` at the beginning, such as reverseVideoGray, reverseVideoTan, reverseVideoBlack, and reverseVideoWhite. Some of the classes named in the style sheets are RHBLUE, RHGREEN, and RHMAGENTA.

Font Some of the style sheets provided by HATS are not named in the templates by default. However, you can apply these style sheets to the templates to change the font family (Arial, Tahoma) or font size of the text. The names of the style sheets give you an idea of their purpose:

- normalFont.css
- scaleableFont.css
- nonFixedFont.css
- largeFont.css
- smallFont.css
- xlargeFont.css
- xsmallFont.css

Unique
Three additional style sheets provided by HATS each have their own unique purpose. These are:

calendar.css
This style sheet controls the appearance of the calendar widget date picker when launched in a new browser window. This style sheet has no effect when the date picker is launched in the current Web page using the **Use inline calendar** setting on the Calendar widget.

inlinecalendar.css
This style sheet controls parts of the appearance of the calendar widget date picker when launched in the current Web page. To display the calendar in the same page, select the **Use inline calendar** setting on the Calendar widget. This style sheet has no effect when the date picker is launched in a new browser window, that is, when not using the **Use inline calendar** setting.

PrintJobWindow.css
This style sheet controls the appearance of a print job in a project, including the PrintJobHeading, ListHeader, and ListEntry.

Stylesheet files are stored in the *project_name*/**Web Content/common/stylesheets** directory. You can edit the stylesheet files by double-clicking on the name of the style sheet in the **HATS Project View** to open the stylesheet editor.

For more information on editing cascading style sheets, see “Using style sheets” in the *HATS User’s and Administrator’s Guide*.

Spreadsheet files (.csv or .xls)

Spreadsheet files in either .csv (comma separated values) or .xls (Microsoft Excel) format can be automatically generated from host screen data defined within the TableWidget. The spreadsheet files are created by the HATS SpreadsheetGeneratorServlet and can be displayed at runtime when the user clicks a defined button or link. A dialog popup displays, and the user can type the directory and file name where the spreadsheet files are to be stored.

For information about creating spreadsheet files, see the Table widget in the *HATS User's and Administrator's Guide*.

Host simulation trace files (.hhs)

Host simulation trace files can be saved and then used to run HATS in a simulated host connection environment instead of using a live host connection. Simulations are created by the Host Simulator Recorder, which acts as a proxy between the real host and the HATS terminal. The host simulation trace files are created in XML format with a file extension of .hhs and are stored in the following directories in the **Host Simulations** folder, which is accessed from the **HATS Projects** view:

- Web projects - *Project_name*/Web Content/WEB-INF/profiles/hostsimulations
- EJB projects - *Project_name*/ejbModule/hostsimulations

For information about creating host simulation trace files, see the *HATS User's and Administrator's Guide*.

ComponentWidget.xml

The ComponentWidget.xml file contains the definitions of all the host components and widgets provided with HATS. If you add your own host components or widgets, you will need to update this file. For an explanation and a small sample of the file, see "Registering your component or widget" on page 25. The ComponentWidget.xml file appears as the last item in your project in the **Navigator** view. To edit the file, double-click the file name in the **Navigator** view and select the **Source** tab.

For a description of the contents and use of this file, see "Registering your component or widget" on page 25.

Web Express Logon configuration file (hatswelcfg.xml)

Web Express Logon (WEL) credential mapping is configured in an XML file named hatswelcfg.xml, which is stored in the root directory of the EAR project for HATS servlet projects. To view the source of the WEL configuration, open the WEL editor from the **Security** tab of the Connection Editor, then click on the **Source** tab. However, you should modify the configuration using the **Network Security Plug-in** and **Credential Mapper Plug-ins** tabs of the WEL editor.

<credentialmapper> tag

The <credentialmapper> tag is the enclosing tag for the Web Express Logon configuration. The attribute of the <credentialmapper> tag is:

class Specifies the Java class to use as the WEL Credential Mapper. This value is not editable on the two plug-in tabs of the WEL Editor. Do not change this value unless directed to do so by IBM Service.

<networksecurity> tag

The <networksecurity> tag is a container for zero or one Network Security plug-in tag (<plugin>), and has no attributes.

<cmplugins> tag

The <cmplugins> tag is a container for a number of Credential Mapper plug-in tags (<plugin>), and has no attributes.

<plugin> tag

The <plugin> tag identifies either a Network Security plug-in or a Credential Mapper plug-in, and encloses the tags for the plug-in's parameters. The attributes of the <plugin> tag are:

class Specifies the Java class that contains the plug-in code.

authenticationtype
Specifies the types of hosts for which the plug-in can map credentials, and is used by Web Express Logon to determine which plug-in will handle a request. Valid values are `AuthType_All`, `AuthType_3270Host`, `AuthType_5250Host`, and `AuthType_VTHost`. Multiple values can be used, separated by vertical bar (`|`). This attribute is valid only for a Credential Mapper plug-in, when the <plugin> tag is enclosed by the <cmplugins> tag.

hostmask
Specifies the names of hosts for which the plug-in can map credentials, and is used by Web Express Logon to determine which plug-in will handle a request. The value of this attribute can contain one or more host addresses, with the vertical bar (`|`) to join multiple addresses, and the asterisk (*) as a wildcard at the beginning, end, or both, of a host address. This attribute is valid only for a Credential Mapper plug-in, when the <plugin> tag is enclosed by the <cmplugins> tag.

<param> tag

The <param> tag identifies a plug-in parameter's name and value. The attributes of the <param> tag are:

name Specifies the name of the plug-in parameter. Each plug-in Java class defines the set of parameter names it will accept.

value Specifies the value of the plug-in parameter. If the parameter is defined in the plug-in Java class to be an encrypted parameter, the value here must be encrypted.

Appendix B. 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:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan, Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, 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 might 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 or changes in the product(s) and 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.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

Intellectual Property Dept. for Rational Software
IBM Corporation
5 Technology Park Drive
Westford, MA 01886
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Programming interface information

This Web Application Programmer's Guide contains information on intended programming interfaces that allow the customer to write programs to obtain the services of HATS.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at “Copyright and trademark information” at www.ibm.com/legal/copytrade.shtml.

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

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Glossary

action. A defined task that an application performs on a managed object as a result of an event, such as a host screen matching the screen recognition criteria specified for a screen event. A list of actions is part of the definition of each event.

ADB. See **application data buffer**.

administrative console. The HATS administrative console is a Web-based utility that provides views and functions to manage licenses and connections, set log and trace settings, view messages and traces, and perform problem determination for HATS Web applications.

application. See **HATS application**.

application data buffer. The format of data that is returned by the WebFacing Server for consumption by the WebFacing application.

application event. A HATS event that is triggered by state changes in the application's life cycle. Examples of application events include a user first accessing a HATS application (a Start event), or an application encountering an unrecognized screen (an Unmatched Screen event).

application keypad. A set of buttons or links representing HATS application-level functions. (Contrast with **host keypad**.)

artifact. See **resource**

background connection. Any connection defined in a HATS application other than the default connection. HATS does not transform screens from background connections. (Contrast with **default connection**.)

bidirectional (bidi). Pertaining to scripts such as Arabic and Hebrew that generally run from right to left, except for numbers, which run from left to right.

BMS map. A screen definition file used with Basic Mapping Support in CICS. A BMS map defines a set of fields which are to be displayed as a group by a CICS application

business logic. Java code that performs advanced functions, such as interacting with other applications, databases, or other systems accessible via Java APIs. Business logic is invoked as an action in an application or screen event.

checkin screen. The screen identifying the host screen that should be active for a connection to be considered ready to be returned to the connection pool. If the application is not on the screen specified by the checkin screen, the connection will be discarded or recycled in attempt to return the connection to the host screen specified by the checkin screen. The checkin screen is only meaningful if connection pooling is specified for a connection.

component. A visual element of a host screen, such as a command line, function key, or selection list. HATS applications transform host components into widgets.

connection. A set of parameters used by HATS, stored in an .hco file, to connect to a host application. (See also **default connection** and **background connection**.)

connection pool. A group of host connections that are maintained in an initialized state, ready to be used without having to create and initialize them.

credential mapper. The component of Web Express Logon that handles requests for host credentials, which have been previously authenticated by a network security layer. (See **network security layer**.)

DDS map. Data Description Specification map. These maps define the layout and behavior of the presentation space for IBM i terminal applications.

Debug. For rich client projects, the same as Run, and in addition enables you to:

- Use the display terminal to see host screens as they are navigated while testing your project
- See debug messages in the Rational SDP console

- See changes you make to your project, for example changing the template or a transformation, without having to restart your application
- Modify and test runtime settings, defined in the runtime-debug.properties file, without modifying the settings, defined in the runtime.properties file, that are deployed to the runtime environment
- Step through Java code, such as HATS business logic

Debug on Server. For Web projects, the same as Run on Server, and in addition enables you to:

- Use the display terminal to see host screens as they are navigated while testing your project
- See debug messages in the Rational SDP console
- See changes you make to your project, for example changing the template or a transformation, without having to restart your application on the test server
- Modify and test runtime settings, defined in the runtime-debug.properties file, without modifying the settings, defined in the runtime.properties file, that are deployed to the runtime environment
- Step through Java code, such as HATS business logic

default connection. The connection on which HATS transforms and presents host application screens to the user. Also referred to as **transformation connection**. (Contrast with **background connection**.)

default rendering. The method used by HATS to render parts of the host screen for which no specific transformation is specified.

deploy. To make a HATS application ready for use in a runtime environment. For HATS Web applications, this includes exporting the HATS project as a Java EE application, that is, as an .ear file, and installing it on WebSphere Application Server. For HATS rich client applications, this includes exporting the HATS project as an Eclipse feature and installing it on individual client systems, either as a stand-alone Eclipse application or from an update site to an existing Eclipse runtime environment.

descriptor. See **screen recognition criteria**.

developer. The person who uses HATS Toolkit to develop applications; also application developer or Web developer. (Contrast with **user**.)

Device Runtime Environment (DRE). A package containing other runtime environments, including the J2SE runtime, which is required to run HATS rich client applications in Lotus Expeditor Client V6.2.0 and earlier. The DRE installs into the runtime environment for Lotus Expeditor Client.

display terminal. A terminal window that displays host screens you can use while testing and debugging to observe interactions between a HATS application and a host application at runtime. You can also interact with the host application using host screens in the terminal window.

Eclipse. An open-source initiative that provides ISVs and other tool developers with a standard platform for developing plug-compatible application development tools. Eclipse is available for download from http://www.eclipse.org.

editor. An application that enables a user to modify existing data. In HATS Toolkit, editors are used to customize resources that have been created by wizards.

Enhanced Non-Programmable Terminal User Interface (ENPTUI). Enables an enhanced interface on non-programmable terminals (NPT) and programmable work stations (PWS) over the 5250 full-screen menu-driven interface, taking advantage of 5250 display data stream extensions.

enterprise archive (EAR). A specialized Java archive (JAR) file, defined by the Java EE standard used to deploy Java EE applications to Java EE application servers. An EAR file contains enterprise beans, a deployment descriptor, and Web archive (WAR) files for individual Web applications. (Sun)

Enterprise JavaBeans (EJB). A component architecture defined by Oracle for the development and deployment of object-oriented, distributed, enterprise-level applications. (Oracle)

event. A HATS resource that performs a set of actions based on a certain state being reached. There are two types of HATS events, application events and screen events.

export. To collect the resources of a HATS project, along with the necessary executable code, into an application EAR file (for Web applications) or Eclipse feature (for rich client applications) in preparation for deploying the application.

Extensible Markup Language (XML). A standard metalanguage for defining markup languages that was derived from and is a subset of SGML.

GB18030. GB18030 is a new Chinese character encoding standard. GB18030 has 1.6 million valid byte sequences and encodes characters in sequences of one, two, or four bytes.

global rule. A rule defining how the rendering of specific input fields should be modified based on certain criteria. Global rules are used in customized screens and screens rendered using default rendering. Global rules can be defined at the project level or at the screen event level.

global variable. A variable used to contain information for the use of actions. The values of global variables can be extracted from a host screen or elsewhere, and can be used in templates, transformations, macros, Integration Objects, or business logic. A global variable can be a single value or an array, and it can be shared with other HATS applications sharing the same browser session.

HATS. See **Host Access Transformation Services**.

HATS application. An application that presents a version of a host application to users, either as a Web-enabled application deployed to WebSphere Application Server, a portlet deployed to a WebSphere Portal, or as an Eclipse client-side processing plug-in deployed to an Eclipse rich client platform such as Lotus Notes or Lotus Expeditor Client. A HATS application is created in HATS Toolkit from a HATS project and deployed to the applicable environment. The deployed application might interact with other host or e-business applications to present combined information to a user.

HATS EJB project. A project that contains the HATS EJB and Integration Objects that other applications can use to get host data. A HATS EJB project does not present transformed screens from a host application.

HATS entry servlet. The servlet that is processed when a user starts a HATS Web application in a browser.

HATS project. A collection of HATS resources (also called artifacts), created using HATS Toolkit wizards and customized using HATS Toolkit editors, which can be exported to a HATS application.

HATS Toolkit. The component of HATS that runs on Rational SDP and enables you to work with HATS projects to create HATS applications.

Host Access Transformation Services (HATS). An IBM software set of tools which provides Web-based access to host-based applications and data sources.

host component. See **component**.

host keypad. A set of buttons or links representing functions typically available from a host keyboard, such as function keys or the Enter key. (Contrast with **application keypad**.)

host simulation. Host simulation enables you to record host simulation trace files that can be saved and then used instead of a live host connection. The recorded trace files can be played back to create screen captures, screen events, and screen transformations using the host terminal function, create and test macros using the host terminal function, test HATS applications using the Rational SDP local test environment, and, along with other traces and logs, aid in troubleshooting a failing scenario in a runtime environment.

host simulation trace. Host simulation trace files record host screens and transactions that can be saved and played back later instead of using a live host connection. Trace files can be recorded using the host terminal function or while in the runtime environment.

host terminal. A HATS Toolkit tool. A session tied to a particular HATS connection, which the HATS developer can use to capture screens, create screen customizations, and record macros.

HTML. Hypertext Markup Language.

HTML widget. See **widget**

Integration Object. A Java bean that encapsulates an interaction with a host screen or a series of host screens. Integration Objects are constructed from macros and can be included in traditional (WSDL-based) Web services, RESTful Web services, or HATS EJB projects. Integration Objects cannot be used in rich client platform applications.

interoperability. The ability of a computer or program to work with other computers or programs.

interoperability runtime. Common runtime used by a combined HATS/WebFacing application to provide management of common connection to the backend host. This runtime decides whether data being returned by the WebFacing server should be handled by the HATS or WebFacing part of the application.

Java Platform, Enterprise Edition (Java EE). An environment for developing and deploying enterprise applications, defined by Oracle. The Java EE platform consists of a set of services, application programming interfaces (APIs), and protocols that provide the functionality for developing multitiered, Web-based applications. (Oracle)

JavaServer Faces (JSF). A framework for building Web-based user interfaces in Java. Web developers can build applications by placing reusable UI components on a page, connecting the components to an application data source, and wiring client events to server event handlers. (Oracle)

JavaServer Pages (JSP). A server-side scripting technology that enables Java code to be dynamically embedded within Web pages (HTML files) and run when the page is served, returning dynamic content to a client. (Oracle)

JavaServer Pages Standard Tag Library (JSTL). A standard tag library that provides support for common, structural tasks, such as: iteration and conditionals, processing XML documents, internationalization, and database access using the Structured Query Language (SQL). (Oracle)

JSF. See **JavaServer Faces**.

JSP. See **JavaServer Pages**.

JSR 168. The Java Portlet Specification addresses the requirements of aggregation, personalization, presentation, and security for portlets running in a portal environment. Version 1.0 of the Java Portlet Specification, Java Specification Request 168 (JSR 168), defines standards to enable portlet compatibility between portal servers offered by different vendors. See **JSR 286**.

JSR 286. The Java Portlet Specification addresses the requirements of aggregation, personalization, presentation, and security for portlets running in a portal environment. Version 2.0 of the Java Portlet Specification, Java Specification Request 286 (JSR 286), defines standards to extend the capabilities of Version 1.0 (JSR 168) to include coordination between portlets, resource serving, and other advanced features. See **JSR 186**.

JSTL. See **JavaServer Pages Standard Tag Library**.

keyboard support. The ability for a developer to enable a user to use a physical keyboard to interact with the host when the application is running in a Web browser or rich client environment. The developer also decides whether to include a host keypad, an application keypad, or both, in a project. If keypads are included, the developer decides which keys are included and how those keys and the keypad appear in the client interface.

keypad support. The ability for a developer to enable a user to interact with the host as if the physical keys on a keyboard were pressed, or to perform tasks related to the application, such as viewing their print jobs or refreshing the screen. See also **application keypad** and **host keypad**.

linked HATS/WebFacing project. A project created by linking a single HATS Web project with a single WebFacing project for the purpose of creating an enterprise application that includes a HATS Web application interoperating with a WebFacing application and sharing a connection to a 5250 backend host.

Lotus Expeditor Client. A standalone client of the Lotus Expeditor product. It is installed on a user or development machine.

Lotus Notes Client. A standalone client of the Lotus Notes product. It is installed on a user or development machine.

macro. A macro, stored in a .hma file, automates interactions with the host. It can send commands to the host, enter data into entry fields, extract data from the host, and be used to navigate screens on behalf of the user.

Model 1 Web pages. A single JSP that contains the information to be presented to the user, formatting tags that specify how the information is displayed, and logic that controls the order in which pages are displayed. (Contrast with **Struts Web pages**.)

network security layer. Software that is responsible for authenticating users and authorizing them to access network resources, such as IBM Tivoli Access Manager.

Operator Information Area (OIA). OIA is the area at the bottom of the host session screen where session indicators and messages appear. Session indicators show information about the workstation, host system, and connectivity.

perspective. In the Rational SDP workbench, a group of views that show various aspects of the resources in the workbench. The HATS perspective is a collection of views and editors that allow a developer to create, edit, view, and run resources which belong to HATS applications.

pooling. See **connection pool**.

portal. An integrated Web site that dynamically produces a customized list of Web resources, such as links, content, or services, available to a specific user, based on the access permissions for the particular user.

print support. The ability for a developer to specify a printer session to be associated with a host session, and enable the user to view host application print jobs, send them to a printer, or save them to disk. Print support is available only for the default connection

Profile. For rich client projects, the same as Run, and in addition enables you to locate the operations that require the most time, and identify actions that are repeated, to eliminate redundancy. You can use this function for performance analysis, helping you to get a better understanding of your application.

Profile on Server. For Web projects, the same as Run on Server, and in addition enables you to locate the operations that require the most time, and identify actions that are repeated, to eliminate redundancy. You can use this function for performance analysis, helping you to get a better understanding of your application.

project. A collection of HATS resources (also called artifacts) that are created using HATS Toolkit wizards and customized using HATS Toolkit editors. These resources are exported as a HATS application. Types of HATS projects include Web, portlet, EJB, rich client, and for purposes of administering HATS Web (including portlet and EJB) applications, HATS administrative console projects. See **HATS project** or **HATS EJB project**.

Rational Software Delivery Platform (Rational SDP). A family of IBM software products that are based on the Eclipse open-source platform and provide a consistent set of tools for developing e-business applications.

rendering set. A rendering set is configured by creating a prioritized list of rendering items. Each rendering item defines a specific region in which a specified host component is recognized and then rendered using a specified widget.

resource. Any of several data structures included in a HATS project. HATS resources include templates, screen events, transformations, screen captures, connections, and macros. Other Rational SDP plug-ins sometimes call these "artifacts."

RESTful Web service. See **Web service, RESTful**.

rich client. A plug-in designed to run on the Eclipse Rich Client Platform in a client environment, and designed to provide an enhanced user experience by the appearance and behavior native to the platform on which it is deployed.

Run. For rich client projects, a function in Rational SDP that enables you to test your HATS rich client projects in an Eclipse, Lotus Notes, or Lotus Expeditor Client instance. In this mode you can modify and test the runtime settings, defined in the runtime.properties file, that are deployed to the runtime environment. Be aware that any changes made to the runtime settings while testing in this mode are retained and become effective when you deploy the HATS application to a runtime environment.

Run on Server. For Web projects, a function in Rational SDP that enables you to test your HATS Web and portlet projects in a WebSphere Application Server as appropriate. In this mode you can modify and test the runtime settings, defined in the runtime.properties file, that are deployed to the runtime environment. Be aware that any changes made to the runtime settings while testing in this mode are retained and become effective when you deploy the HATS application to a runtime environment.

runtime settings. Log, trace, and problem determination settings defined in the runtime.properties file that are deployed to the runtime environment.

screen capture. An XML representation of a host screen, stored in a .hsc file, used to create or customize a screen customization, screen combination, transformation, global rule, or macro. Screen captures are useful because they enable you to develop a HATS project even when not connected to the host. They are also useful in creating macros which are the core of HATS Integration Object and Web services support.

Screen captures of video terminal (VT) host screens can be used to create or customize a macro using the Visual Macro Editor and as the check-in screen when configuring pooling. They cannot be used to create screen customizations, screen combinations, transformations, default rendering, or global rules.

screen combination. A type of HATS screen event designed to gather output data from consecutive, similar host screens, combine it, and display it in a single output page. The screen combination definition, stored in a .evnt file, includes a set of screen recognition criteria for both the beginning and ending screens to be combined, how to navigate from screen to screen, and the component and widget to use to recognize and render the data gathered from each screen.

screen customization. A type of screen event designed to perform a set of actions when a host screen is recognized. A screen customization definition, stored in a .evnt file, includes a set of criteria for matching host screens, and actions to be taken when a host screen matches these criteria.

screen event. A HATS event that is triggered when a host screen is recognized by matching specific screen recognition criteria. There are two types of screen events, screen customizations and screen combinations.

screen recognition criteria. A set of criteria that HATS uses to match one or more screens. When a host displays a screen, HATS searches to determine whether the current host screen matches any of the screen recognition criteria defined for any screen event in your project. If HATS finds a match, the defined actions for the screen event are performed.

Screen recognition criteria are also used in the process of recording a macro; in this context they are sometimes called **descriptors**.

Secure Sockets Layer (SSL). A security protocol that provides communication privacy. SSL enables client/server applications to communicate in a way that is designed to prevent eavesdropping, tampering, and message forgery. SSL was developed by Netscape Communications Corp. and RSA Data Security, Inc.

source. The files containing the markup language that define a HATS project or one of its resources. Also the name of a folder contained in each HATS project.

SSL. See **Secure Sockets Layer**.

standard portlets. Portlets that comply with the standard portlet APIs defined by Java Portlet Specifications JSR 168 or JSR 286. See **JSR 168** and **JSR 286**. .

Standard Widget Toolkit (SWT). An Eclipse toolkit for Java developers that defines a common, portable, user interface API that uses the native widgets of the underlying operating system.

Struts Web pages. Web pages built using the Apache Software Foundation's Struts open-source framework for creating Java web applications. This method of building Web pages creates class files that set values and contain getters and setters, input and output JSPs, and a Web diagram to display the flow and logic of the Web pages. (Contrast with **Model 1 Web pages**.)

SWT. See **Standard Widget Toolkit**.

system screen. An IBM i screen for which data description specification (DDS) display file source members are not available. System screen is specific to an application on an IBM i platform that has been WebFaced.

template. A template, stored in a .jsp file (for Web projects) or a .java file (for rich client projects), controls the basic layout and style, such as color and font, of the application. It also defines the appearance of areas that are common in your GUI, such as a banner and a navigation area.

text replacement. A HATS function used to transform text on a host system into images, HTML code, or other text on a HATS screen transformation,

theme. A theme groups a set of common appearance and behavior characteristics for a project. These attributes can be individually modified later.

transfer. To copy an application EAR file to the server, typically by FTP.

transformation. A transformation stored in a .jsp file (for Web projects) or a .java file (for rich client projects) defines how host components should be extracted and displayed using widgets in your GUI.

transformation connection. See **default connection**.

transformation fragment. A HATS resource that contains the content with which to replace all occurrences of a pattern in any given transformation.

Unicode. A universal character encoding standard that supports the interchange, processing, and display of text that is written in any of the languages of the modern world. It also supports many classical and historical texts in a number of languages. The Unicode standard has a 16-bit international character set defined by ISO 10646.

user. Any person, organization, process, device, program, protocol, or system that uses the services of a computing system.

user list. A list containing information about accounts (user IDs) that a HATS application can use to access a host or database. User lists contain user IDs, passwords, and descriptions for the accounts.

UTF-8. Unicode Transformation Format, 8-bit encoding form, which is designed for ease of use with existing ASCII-based systems.

Web archive (WAR). A compressed file format, defined by the Java EE standard, for storing all the resources required to install and run a Web application in a single file.

Web Express Logon (WEL). A HATS feature that enables users to log onto several hosts using a set of credentials that are authenticated by a network security layer. (See **network security layer**.)

Web service. A self-contained, self-describing modular application that can be published and invoked over a network using standard network protocols.

Web service, RESTful. A Web service that uses a stateless architecture and is viewed as a resource rather than a function call. Well-formatted URIs are used to identify the Web service resource, HTTP method protocols are used to do create, retrieve, update, and delete (CRUD) activities, and HTTP header information is used to define the message format.

Web service, traditional, WSDL-based. A Web service where typically, XML is used to tag data, SOAP is used to transfer data, WSDL is used for describing the services available, and UDDI is used for listing what services are available.

WebFacing feature. The IBM WebFacing Tool for IBM i feature of the HATS Toolkit. The WebFacing feature provides the ability to convert IBM i data description specification (DDS) display file source members into a Web-based user interface for existing 5250 programs.

WebFaced application. A Web application produced by the WebFacing feature of the HATS Toolkit.

WebSphere. An IBM brand name that encompasses tools for developing e-business applications and middleware for running Web applications. Sometimes used as a short name for WebSphere Application Server, which represents the runtime half of the WebSphere family of products.

WebSphere Application Server. Web application server software that runs on a Web server and that can be used to deploy, integrate, run, and manage e-business applications. HATS applications, when exported and transferred to a server, run as WebSphere Application Server applications.

WEL. See **Web Express Logon**.

widget. A reusable user interface component such as a button, scrollbar, control area, or text edit area, that can receive input from the keyboard or mouse and can communicate with an application or with another widget. HATS applications transform host components into widgets.

wizard. An active form of help that guides users through each step of a particular task.

workbench. The user interface and integrated development environment (IDE) in Eclipse-based products such as Rational SDP.

XML. See **Extensible Markup Language**.

Various Java definitions reprinted with permission from Oracle.

Index

Special characters

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W

X

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

IBM Host Access Transformation Services
Web Application Programmer's Guide
Version 9.5

Publication No. SC27-5902-01

We appreciate your comments about this publication. Please comment on specific errors or omissions, accuracy, organization, subject matter, or completeness of this book. The comments you send should pertain to only the information in this manual or product and the way in which the information is presented.

For technical questions and information about products and prices, please contact your IBM branch office, your IBM business partner, or your authorized remarketer.

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. IBM or any other organizations will only use the personal information that you supply to contact you about the issues that you state on this form.

Comments:

Thank you for your support.

Submit your comments using one of these channels:

- Send your comments to the address on the reverse side of this form.
- Send a fax to the following number: 1-800-227-5088 (US and Canada)
- Send your comments via email to: USIB2HPD@VNET.IBM.COM

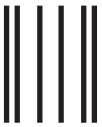
If you would like a response from IBM, please fill in the following information:

Name	Address
Company or Organization	
Phone No.	Email address

Readers' Comments — We'd Like to Hear from You
SC27-5902-01

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
Rational Enterprise Modernization UAD
Department 67RA/Building 503
Research Triangle Park, NC 27709-9990

Fold and Tape | **Please do not staple** | Fold and Tape

SC27-5902-01

Cut or Fold
Along Line

Printed in USA

SC27-5902-01