# IBM® WebSphere® Host Access Transformation Server Developer's Guide

*Version 4*

SC31-6324-00

# IBM® WebSphere® Host Access Transformation Server Developer's Guide

*Version 4*

SC31-6324-00

**Note**

Before using this information and the product it supports, be sure to read the general information under Appendix D, "Notices" on page 159.

**First Edition (December 2002)**

# Contents

# Preface

The ***HATS Getting Started*** manual explains how to create HATS projects using the wizards in the HATS Studio. The information in ***HATS Developer's Guide*** is designed to help you, the Host Access Transformation Server (HATS) developer, understand how to modify HATS projects using the HATS Studio editors.

## Where can I find information about HATS?

You can find the following documentation online after you have installed HATS:

- PDF versions of ***HATS Getting Started*** and ***HATS Developer's Guide*** are accessible from the Windows Start menu.
- HTML versions of both books are accessible from within the WebSphere Studio Help menu, and also from the Windows Start menu.
- The ***HATS Readme file*** is accessible from the Windows Start menu.
- Context-sensitive help is available on all fields in the HATS wizards and editors. Press the F1 key to see the help.
- Tips are provided at key points in the process of developing a HATS project. You can control whether you want to continue to see tips.

To view the documentation before you install HATS, insert the HATS CD and choose **View Documentation** from the Welcome screen.

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

# Chapter 1. Using Host Access Transformation Server (HATS)

Suppose you want to make all of your host applications available on the Web. Suppose that you also want to create a host application that provides the look, feel, and easy navigation of a Web page, enabling end users to access host application functions using buttons, drop-down lists, links, and option lists. Host Access Transformation Server (HATS) not only enables you to customize host applications to work in this manner, it enables you to customize them without programming.

Using HATS, you can wrap your host application "green screen" with Web-style borders and add headings, company logos, and links to other Web sites. Global text replacement on the host screen enables you to give your application a custom feel. Users can skip unnecessary host screens and be prompted for input. You can also choose to present any host screen element (such as host application function keys) in a fully-interactive, Web-style output (such as links) and define where users will see the output in the application. HATS gives you the control to fully customize your host application and enhance the ways in which users work with applications.

HATS macro support enables you to provide programmed navigation through multiple host screens. For example, a macro can navigate the screens of a host application to display the first screen the end user needs to see, bypassing all the screens in between. You can combine data from multiple host screens into a single HATS screen. HATS global variables can extract data from host screens and enter the data on other screens. You can use business logic to integrate legacy systems with other back-end systems in your company, as well as your business partners' systems. You can also use business logic to perform complex calculations and automatically enter the results into host forms.

The foundation of HATS is the HATS Studio. The HATS Studio provides you with all of the necessary tools to create, assemble, and transfer host applications to the production system for deployment. The resulting host application can be accessed by end users with standard Web browsers.

## Understanding HATS application processing

Before creating a HATS project, you should understand how HATS processes host applications. As users access each screen of an application, HATS processes the application as described in the following steps. Figure 1 shows the flow of these steps. Each *italicized* term in the following steps is a key concept of HATS. Key concepts are described in "Understanding HATS key concepts and objects" on page 4.

1. When the host displays a screen, HATS compares the host screen to each *screen recognition criterion* defined in the project's enabled *screen customizations*, in the order defined by *event* priority, until a match is found.
2. When matched, HATS performs the *actions* defined for the screen customization. These actions can include:
   - Applying a *transformation* using the associated *template*. HATS displays any *host components* (defined in the transformation) as HTML *widgets*.
   - Executing *business logic*
   - Interacting with *global variables*

- Showing a URL
- Playing a *macro*.

3. If no screen recognition criteria match the host screen, HATS processes the unmatched screen event. The default action of this event is to display the host screen using the default transformation and applying the *default template*.
4. As the host presents each new screen of an application, HATS begins at Step 1 again and proceeds through these steps.

   **Note:** If a macro uses skip-screen processing, those screens are not subject to these steps.

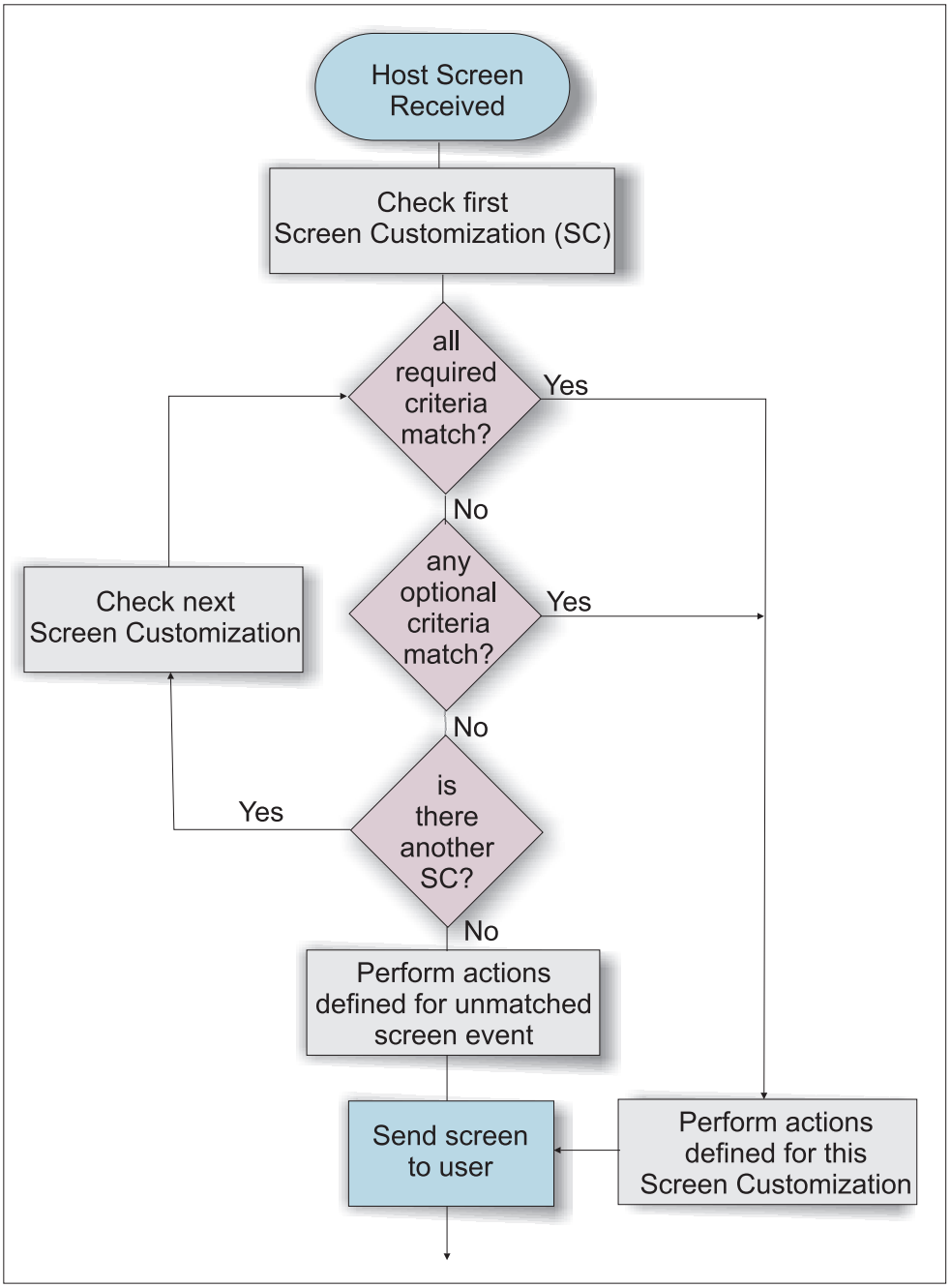



*Figure 1. HATS screen processing*

Screen customizations are an important concept in the development of a HATS project. Without screen recognition criteria being defined in a screen customization, HATS would not know what actions to take when the host screen is encountered. You can obtain the greatest level of customization for your host application by applying transformations and displaying host components.

You should familiarize yourself with the basic principles of screen customization before beginning the development process. The core elements of setting screen recognition criteria are discussed in Chapter 4, “Editing a screen customization” on page 17.

## Understanding HATS key concepts and objects

This section explains key concepts and objects of HATS, some of which are described in "Understanding HATS application processing" on page 1. Many key objects in HATS are created using a wizard, but viewed or modified after creation using an editor.

**Project**
A collection of HATS resources, created using wizards in HATS Studio and customized using HATS Studio editors, that are assembled into a HATS application.

**Event**
A list of actions performed when the application has reached a certain state. There are four types of events in HATS applications:

- Connect
- Disconnect
- Unmatched screen
- Matched screen

The first three events are defined by HATS, and you can locate them in the `source/profiles/events/session/main` directory path of the **Navigator** tab of the HATS Studio. If you want to modify these events for any reason, you can edit them by double-clicking on the event. The matched screen event is also known as a screen customization.

**Screen customization**
A HATS resource, an event, with two parts: a set of screen recognition criteria, and a list of actions to be taken.

**Screen recognition criteria**
During project creation, you set screen recognition criteria that HATS uses to match host screens. Host screens can be recognized by any combination of criteria including how many input fields or total fields are on the screen, the coordinates of the cursor's position, and text strings on the screen within a defined rectangle or anywhere on the screen.

When a host displays a screen, HATS searches to see whether the current host screen matches any of the screen recognition criteria you set in any screen customizations in your project. If HATS finds a match, the defined actions for the screen customization are performed.

For more information on setting screen recognition criteria, see Chapter 4, "Editing a screen customization" on page 17.

**Action**
A step that occurs when a host screen is encountered that matches the screen recognition criteria specified for a screen customization. A list of actions is part of the definition of each screen customization.

**Transformation**
A transformation is a JSP file that defines how host components should be extracted and displayed in a Web presentation. Applying a transformation is one of the possible actions of a screen customization.

For more information on creating transformations, see Chapter 5, "Editing a transformation" on page 25.

**Host components**

Host components are HATS objects responsible for recognizing elements of the host screen (such as command lines, function keys, menus) that you choose to present to the end user of the HATS application. You can use the set of host components that HATS supplies, or you can create your own host components. For information on creating custom host components, see "Creating a custom host component" on page 57.

For more information on selecting host components to use with your HATS project, see "Insert Host Component wizard" on page 26.

**Widgets**

Widgets are HATS objects responsible for creating the HTML output for host components in the HATS presentation. For example, you can convert function key host components into button widgets so that the end user sees the function keys as buttons in the HATS application. You can use the widgets that come with HATS, or you can create your own. For information on creating custom widgets, see "Creating a custom widget" on page 58.

For more information on selecting widgets to use with your HATS project, see "Insert Host Component wizard" on page 26.

**Template**

A template is a JSP file that enables you to enhance the appearance of your project. When creating a HATS project, you select a template to use as the default template for your project. The template can contain company logos and information and links to other Web pages. You can select from the sample templates that are provided with HATS, or you can design custom templates for your projects using the wizards and editors in HATS Studio. You can choose which template to apply to the host screen when a transformation is applied.

In a Web page, the template surrounds the area where the transformation appears. For more information on associating a template with a transformation, see Chapter 6, "Using templates" on page 31.

**Business logic**

Any Java code invoked as an action in an event, such as a screen customization. Business logic is specific to the application and is not provided as part of HATS.

For more information about business logic, see Chapter 9, "Adding business logic" on page 41.

**Global variable**

A variable used to store a value that can be used throughout the lifetime of an application. The value of a global variable can be extracted from a host screen or defined by the developer. Global variables can be used in templates, transformations, screen customization actions, macros, or business logic.

For more information about global variables, see Chapter 7, "Interacting with global variables" on page 35.

**Macro**

An XML script that defines a set of screens and defines certain actions that should be taken on those screens. Macros are used to automate end user interactions with the host. You can record and play macros to skip screens, prompt users for data input, and extract host screen information. A macro

that skips screens does not require the user to interact with each screen. Prompt macros request information from the user that is used on other screens, thereby limiting user interaction with the host. An extract macro can be used to retrieve information to present to the user.

A macro can be played as an action of a HATS screen customization. It is the last action defined for a screen customization. For information on playing macros as an action of a screen customization, see "Actions tab" on page 19. For more information on incorporating macros into the HATS environment, refer to Chapter 8, "Incorporating macros" on page 37.

**HATS terminal**
A connection in HATS Studio to a live host. Using the HATS terminal, you can capture screens, create screen customizations and transformations, and record macros. You can also play previously recorded or imported macros.

**Screen capture**
An XML representation of a host screen, used to create or customize a screen customization or transformation.

**Run on Server**
A function in HATS Studio that enables you to run your project on the WebSphere Studio internal WebSphere Application Server and see the output. This function is also known as the WebSphere Test Environment (WTE).

**Print support**
The ability for a developer to specify a printer session to be associated with a host session, and enable the end user to view host application print jobs, send them to a printer, or save them to disk.

For more information on print support, see Chapter 11, "Enabling print support in projects" on page 45.

**Keyboard support**
The ability for a developer to enable an end user to use a physical keyboard to interact with the host when the application is run in a Web browser. The developer also decides whether to include a host keypad, an application keypad, or both, in a project. If the keypads are included, the developer decides which keys are included and how those keys and the keypad appear in the Web browser.

The host keypad is a table of buttons or links that enable the end user to interact with the host as if they pressed the physical keys on a keyboard. However, the end user can still use the physical keys on the keyboard instead of the buttons or links on the host keypad.

The application keypad is a table of buttons or links that enable the end user to perform tasks related to the application, such as viewing their print jobs or refreshing the screen.

For more information on keyboard support, see Chapter 12, "Enabling keyboard support in projects" on page 49.

# Chapter 2. Creating and organizing projects

When you create a new HATS project, a set of folders is created to help you organize your HATS files. The highest level folder has the same name as the name you give to your project when you create it. In that folder are other high-level folders that contain objects defined in your HATS project:

- Project settings
- Screen Customizations
- Transformations
- Templates
- Screen Captures
- Macros
  - Macro Event Handlers
- Source
- Common
  - Images
  - Stylesheets

How to work with the contents of each of these folders is explained in this book.

You can create folders within these high-level folders to help organize your project. For instance, as you create screen captures for your project, you might want to create folders under the Screen Captures folder to organize and group the captured screens. To create a folder, right-click on one of the high-level folders in the tree and select **New HATS > Folder**. To move a file into a different folder, right-click on the file and select move, or you can use the drag-and-drop method. You can create folders under any of the high-level folders to help organize your files.

HATS projects, created in HATS Studio, are extensions of Web projects in the WebSphere Studio workbench. For more information about Web projects, open the Help perspective in the WebSphere Studio workbench and select **Application Developer Documentation**. Expand the sections as follows to find information on Web projects: **Concepts > Projects > Web projects**.

By default, all HATS applications are stored in one Enterprise Archive file, HATS.ear. When you assemble your applications and deploy them on WebSphere Application Server (WAS), the HATS.ear file contains a Web Archive (.war) file with the resources to run each application, as well as one copy of the HATS run-time executable code. If you prefer, you can organize your applications differently, either each in its own .ear file, or in some other combination.

**Note:** If your .ear file contains the .war files for multiple applications, which is the default setting, your WebSphere Application Server (WAS) administrator must use WAS administration to configure the server with **Module Visibility** set to **Application**. If the server is not configured this way, your applications will not run. Your end users will get the following error message in their browser: `HTTP 404 - File not found` when trying to run an application.

Which .ear file your project files go into is determined when you create the project. On the first panel of the Create a HATS Project wizard, you can keep the **Use**

**default Enterprise Application project** check box checked, in which case project files are created within HATS.ear, or you can clear the check box and specify a different .ear file to contain your project's files. In HATS Studio, you cannot choose a different .ear file after the project is created. However, you can move projects from one .ear to another, using the WebSphere Studio application.xml editor.

To move a project, follow these steps:

1. Click the **Navigator** tab of the HATS Studio to display the .ear files.
2. Expand the .ear file to which you want to move a project. Expand the META-INF folder and locate the application.xml file.
3. Start the WebSphere Studio application.xml editor by double-clicking the application.xml file.
4. In the application.xml editor, click the **Modules** tab to display the .war files for your projects.
5. Click **Add** to display the **Folder Selection** dialog.
6. Select the project you want to add to the .ear file.

   **Note:** Make sure you select the project, and not the .ear file with the same name.
7. Click **OK**.

If you want to move a project to another machine with WebSphere Studio installed, you must export the project as a .zip file before sending the project to another machine. The following steps explain how to export a project and import it into a WebSphere Studio installed on another machine.

To export the project from WebSphere Studio to a .zip file:

1. Highlight the project in the **HATS Project View** tab of the HATS Studio.
2. Select **File > Export** to open the Export wizard.
3. Select **Zip file**, and click **Next**.
4. Check the boxes for both the project name file and the project .ear file.
5. In the **Zip file** field, type in the destination to where you want to export the file. You can also select an export destination by clicking **Browse**.
6. Click **Finish**.

To import the project in a .zip file into WebSphere Studio:

1. Create a new HATS project.
2. Select **File > Import** to open the Import wizard.
3. Select **Zip file**, and click **Next**.
4. In the **Zip file** field, type in the location of the .zip file. You can also select the location by clicking **Browse**.
5. Click **Browse** next to the **Folder** field, and select the name of the new HATS project you created.
6. Click **Finish**.
7. When the .zip file is being imported, click **Yes to all** to overwrite existing files.

The new project will contain the original HATS project.

Consider the effect of the following on your server machine when deciding how to arrange your projects:

**Disk space**

If you create each project in its own .ear file, it has its own copy of the HATS run-time code, of which there is one copy per .ear file. The run-time code is approximately 16 MB, so you can multiply that by the number of projects you have to see how much disk space is consumed on your WAS machine for each project.

**Maintenance**

If you update a project and re-deploy it, you are re-deploying all the projects in that ear file.

**Applying service**

If an update is required to the HATS run-time executable code, it must be applied to each copy of the run-time code in each .ear file.

**Logging and tracing**

Logging and tracing are controlled at the level of the .ear file, not at the individual HATS application level. If each HATS application is in its own .ear file, you can control its logging and tracing settings independently of any other applications. If you have several HATS applications in one .ear file, logging and tracing settings apply to all HATS applications in the .ear file. Messages for all HATS applications in the .ear file are inserted into the same log file, and trace information for all HATS applications is inserted to the same trace file.

See Chapter 16, "Troubleshooting HATS" on page 65 for more logging and tracing information.

**License tracking**

License tracking is also controlled at the level of the .ear file, not at the individual HATS application level. If each HATS application is in its own .ear file, license tracking is done independently of other applications. If you have several HATS applications in one .ear file, license tracking is performed for all HATS applications in the .ear file. Information about license usage is kept for all HATS applications in the .ear file, and is inserted into the same license usage file.

See Chapter 16, "Troubleshooting HATS" on page 65 for more information on license tracking.

# Chapter 3. Modifying a HATS project

When you created your project using the HATS Studio Create a Project wizard, the settings you chose on the panels were saved in a project application (.hap) file. You can use the project editor to view and modify those settings. You can invoke the project editor by double-clicking on the **Project Settings** node under the name of the project you want to modify in the **HATS Project View** tab of the HATS Studio.

The settings displayed in the project editor are the settings that are used for the entire project. If you want to modify any of the settings, you can use the tabs of the project editor to do so. Changes made in the project editor are automatically recognized when using **Run on Server** in the HATS Studio by clicking **Refresh** on the application keypad or by displaying a new host screen in the browser.

The following sections describe each tab of the project editor, and explain how they can be used to modify the project settings.

## Overview tab

The **Overview** tab of the project editor summarizes all of the settings you specified when you created your project. The only item you can modify on this tab is the description of your project.

Each of the section headings on the **Overview** tab is a link to the other tabs of the project editor.

## Connection Settings tab

The **Connection Settings** tab displays the following items:

- The name of the host to which your project connects. This is either the name or IP address of the Telnet server.
- The number of the port through which the project connects. Typically, port 23 is used, but the administrator of the Telnet server may have modified the port number.
- The type of host session for the project, such as TN3270, TN3270E, or 5250.
- The code page used for the project. Select the codepage for your country.
- The screen size of the host session. The screen size defines the number of rows and columns in the host screen.
- Whether SSL is enabled for the project.

The **host name** and **port number** fields are entry fields. You can change the name of the host to which your project connects or the port number through which your project connects by typing over the name and number that appear in these fields.

The type of **session**, **code page**, and **screen size** fields have drop-down lists from which you can select certain values for the fields.

The **SSL enabled** checkbox allows you to enable or disable SSL by checking or clearing the checkbox. If SSL is enabled, you can import the required certificate file using **Import**. Refer to Chapter 13, “Enabling SSL security” on page 53 for more information.

**Notes:**

1. The values for screen size change depending on the code page and type of host session you select.
2. If you select a bi-directional (BIDI) code page, refer to "Enabling the user to reverse the screen direction" on page 84.

## Advanced Connection Settings tab

The **Advanced Connection Settings** tab enables you to specify settings for print support as well as any additional IBM WebSphere Host On-Demand connection session parameters you want to use.

Click the **Enable print support** checkbox if you want print support.

For 3270 hosts, if you check the **Enable print support** checkbox, you can choose the paper size, page orientation, and the print font you want to use for printing jobs from your project from the drop-down lists for each parameter. Refer to "Defining print support for your project" on page 45 for more information.

For 5250 hosts, if you check the **Enable print support** checkbox, you must specify the URL of the iSeries™ for Web Access (IWA) Printer Output window. The default URL is http://*hostname*/webaccess/iWASpool, where *hostname* is the name of the 5250 host. The end user of your application can set print options in the IWA Printer Output window. Refer to "Defining print support for your project" on page 45 for more information.

You can add, modify, or remove any IBM WebSphere Host On-Demand session parameters using the buttons to the right of the table of parameters. If you click **Add**, you can select a parameter using the drop-down list next to the **Name** field. For a list of the valid parameters that you can add and their descriptions, see "<otherParameters> tag" on page 110.

## Template tab

The **Template** tab displays the template used to surround a transformation when applying a transformation is the action of a screen customization in your project. On this tab, you can change which template to use as the default template.

The default template is the template applied to all transformations in the project, and the template applied with the default transformation as the default action of the unmatched screen event. For infomation on how to modify the action of the unmatched screen event, see "Screen customization ordering" on page 23.

When creating or modifying the actions of a screen customization, you can override the default template chosen by selecting a different template.

## Text Replacement tab

HATS applications can convert text strings on host screens into different strings on Web pages. These text strings must be protected text strings; that is, strings that are used only on the screen and never passed to the host application as input. For example, you can change the text string used as a field label or prompt, but not the text in an entry field.

The **Text Replacement** tab displays a table in which you can specify any text you want to replace, the text with which you want to replace the original text, and whether the texts are case sensitive.

**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 could be contracted, expanded, or forced to a new line.

You can add, modify, or remove any text replacement specifications by using the buttons to the right of the table of values.

**Note:** If you select a bi-directional (BIDI) code page, refer to "Additions to HATS files" on page 86.

## Event Priority tab

The **Event Priority** tab displays a list of the screen customizations contained in your project.

If the checkbox next to the name of a screen customization is checked, that screen customization is enabled for the project. When a screen customization is enabled, and the screen recognition criteria match the host screen, HATS performs the actions specified for that screen customization, and no more screen customizations are checked for matches. When a screen customization is disabled, HATS ignores the screen customization. If you want to test certain screen customizations, you might want to disable other screen customizations. If so, clear the checkbox while you are testing.

HATS applications check each incoming host screen against the list of screen customizations. If there are multiple screen customizations that match a given screen, the first screen customization that matches the screen is applied. The higher priority screen customizations should be near the top of the list. For example, you might have one screen customization that recognizes a few specific screens, and a second one that recognizes a more general set of screens. If the second screen customization is higher in the list than the first, a screen might be recognized by the more general screen recognition criteria and perform the associated actions, rather than recognizing the screen by the more specific criteria and performing the associated actions of the first screen customization.

If you want to change the priority of any of the screen customizations, highlight the screen customization by clicking it. Then click either **Up** or **Down** to move the screen customization higher or lower in the list.

For more information, see "Screen customization ordering" on page 23.

## General tab

The **General** tab displays the settings for host components, widgets, keypads, keyboard support, and client locale.

You can customize the default project settings for each of these by clicking on the node in the customization settings tree. To change settings for individual host components and widgets, you must expand them in the tree to see the individual host components and widgets. Some of the host components and widgets do not

have any customizable settings. For information on the settings that can be customized with the Insert Host Component wizard, see “Component and widget settings” on page 89.

You can override the default project settings for host components and widgets when you insert them into transformations. Those modified settings only apply to the individual instances of those host component or widget in the transformation. All the other instances of the host component or widget for any transformation in the project still use the default project settings, unless you modify them. For example, you have default settings for the VisualTable component. In a single transformation, you may have two VisualTable components; one that uses the default settings from the project settings, and another that uses modified settings.

## Application keypad

You can customize the following settings for the application keypad:

**Show default application keypad**
Click the checkbox if you want a default application keypad defined in templates or transformations to be displayed in the HTML output when users interact with your application.

**Select keys to display**
If the **Show default application keypad** checkbox is checked, you can click the checkboxes next to each of the keys that you want to include on the default application keypad in the HTML output.

**Display as**
Select the value from the drop-down list to determine whether the selected keys display as buttons or links.

## Host keypad

You can customize the following settings for the host keypad:

**Show default host keypad**
Click the checkbox if you want a default host keypad defined in templates or transformations to be displayed in the HTML output when users interact with your application.

**Select keys to display**
If the **Show default host keypad** checkbox is checked, you can click the checkboxes next to each of the keys that you want to include on the default host keypad in the HTML output. If you want to include all the available keys, you can click **Select all**. You can click **Deselect all** to clear all of the boxes that are checked.

**Display as**
Select the value from the drop-down list to determine whether the selected keys display as buttons or links.

## Keyboard support

You can customize the following settings for keyboard support:

**Turn keyboard support on**
Click the checkbox if you want your end users to be able to use the physical keyboard keys to interact with the host. This enables the end user to click representations of host aid keys, such as the function, SYSREQ, or ATTN keys. However, if keyboard support is turned on, the end user will

not be able to use those same keys for other functions, such as requesting help for the browser by pressing the PF1 key.

**Turn initial keyboard state on**
If the **Turn keyboard support on** checkbox is checked, you can click this checkbox to make the keyboard enabled when the HTML output is initially displayed to the user.

### Client locale

You can customize the following settings for the client locale:

**Select where to acquire the language to display button captions and messages**
Click one of the following radio buttons:

**From the browser's "Accept-Language" header**
The language used to display button captions and messages is determined by the language specified by the end user's browser.

**From the server's primary locale**
The language used to display button captions and messages is determined by the locale of the machine where the application is deployed.

**Always use the following language**
You can select the language to use for button captions and messages from the drop-down list.

## Source tab

The **Source** tab displays the tags and values in the `application.hap` file for all the settings you selected or defaulted to in your project. As you make changes on other tabs in the project editor, the tags and values displayed in the source file change to match.

You can also make changes to the tags and values in the source file, and they are reflected on the appropriate tabs of the project editor. For information about the tags in the `application.hap` file, refer to "Application files (.hap)" on page 105.

# Chapter 4. Editing a screen customization

A screen customization is a HATS resource with two parts: a set of screen recognition criteria used to match host screens, and a list of actions to be taken when a host screen matches the screen recognition criteria.

When you created your project in HATS Studio, you used the Create a HATS Screen Customization wizard to define screen customizations. The recognition criteria and the actions you defined for the screen customization were saved in a screen customization (.evnt) file. You can use the screen customization editor to view and modify those criteria and actions.

You can see the screen customizations you have created by expanding the **Screen Customizations** node of the **HATS Project View** tab of the HATS Studio. You can invoke the screen customization editor by double-clicking on the name of the screen customization.

The following sections describe each tab of the screen customization editor.

## Overview tab

The **Overview** tab of the screen customization editor summarizes all of the information you specified when you created your project. It contains the name and description of the screen customization, the name and an image of the screen that was used to create the screen recognition criteria, a summary of the screen recognition criteria, and a summary of the actions to be taken when the screen is recognized. On this tab, you can modify the description of the screen customization, and you can select a different screen to associate with the screen customization. The selected screen is the screen that is used whenever you make changes to the screen customization, such as modifying the screen recognition criteria, or adding actions.

Each of the section headings on the **Overview** tab is a link to the other tabs of the screen customization editor.

## Screen Recognition Criteria tab

During project creation or customization, you set screen recognition criteria that HATS uses to match host screens. Host screens can be recognized by any combination of criteria including how many input fields or total fields are on the screen, the coordinates of the cursor's position, and text strings on the screen within a defined rectangle or anywhere on the screen. The **Screen Recognition Criteria** tab of the screen customization editor displays the screen recognition criteria that you set for the screen customization. You can add, edit, or remove criteria on this tab.

### Field criteria

You can use the total number of fields on a screen, the total number of input fields on a screen, or both as screen recognition criteria. These are the first two criteria shown on the **Screen Recognition Criteria** tab.

If you click the checkbox for these criteria, they are used to recognize the screen. The fields next to each field criterion show the number of fields and input fields for the screen specified on the **Overview** tab. If you change the screen being used for this screen customization, click **Refresh** to update the values for the screen you choose.

**Note:** If you are using field criteria to recognize screens that have a certain number of fields, and another screen does not contain the same number of fields, that screen is not recognized. For example, one screen might have a list of 10 files with 10 fields. If the host displays a screen with only eight files in the list and eight fields, the second screen does not match the number of fields criterion of the screen customization that matched the first screen.

For an explanation of the **Optional** and **Invert** checkboxes, see "Optional versus non-optional screen recognition criteria" on page 19 and "Inverted match of screen recognition criteria" on page 19.

## Cursor position criteria

You can also use the initial position of the cursor as a screen recognition criterion, either by itself or in conjunction with other criteria, by clicking the checkbox. The fields next to the cursor position criterion shows the row and column of the cursor position for the screen specified on the **Overview** tab. If you change the screen being used for this screen customization, click **Refresh** to update the values for the initial cursor position row and column on the screen you choose.

For an explanation of the **Optional** and **Invert** checkboxes, see "Optional versus non-optional screen recognition criteria" on page 19 and "Inverted match of screen recognition criteria" on page 19.

## Text string location criteria

Text string location criteria are shown at the bottom of the **Screen Recognition Criteria** tab. If you have set a string location as a screen recognition criterion, it is shown in the table. The table shows what part of the screen contained the string, shows some of the characters of the text selected, and whether the text is case sensitive.

If you highlight a row of the table and click **Edit**, or if you click **Add**, the **Screen Recognition Criterion** dialog appears. In the dialog panel, you can either modify or specify text string information. The panel shows the screen selected on the **Overview** tab.

You can select any text on the screen by drawing a rectangle around the text. Place your cursor at any point on the screen, click and hold the left mouse button, and move the cursor to another location on the screen to draw the rectangle. The fields on the right of the dialog show the text you selected and the starting and ending row and column numbers of the rectangle. You can specify the part of the screen that should contain the text by clicking one of the radio buttons for **Anywhere on the screen**, **At a specified position**, or **Within a rectangular region**. If the text you selected must be case-sensitive to be recognized as matching the screen recognition criteria, click the **Case sensitive** checkbox.

For an explanation of the **Optional** and **Invert** checkboxes, see "Optional versus non-optional screen recognition criteria" on page 19 and "Inverted match of screen recognition criteria" on page 19.

Click **OK** when you have finished your selections.

## Optional versus non-optional screen recognition criteria

You can choose whether the screen recognition criteria you set is optional or non-optional. If you do not check the **Optional** checkbox, the recognition criterion is considered non-optional. How you use the **Optional** checkbox corresponds to the Host On-Demand screen descriptor attribute, **optional**.

If you have both optional recognition criteria and non-optional recognition criteria, HATS checks the non-optional criteria first. If all the non-optional criteria match, the screen matches. If at least one of the non-optional criteria does not match, HATS checks the optional criteria. For a screen to match the criteria, HATS must find all non-optional criteria, or at least one optional criterion. Otherwise, the screen fails to match. The following example explains this concept in greater detail.

**Note:** Non-optional does not mean required.

Suppose you defined cursor position location and two text strings with the values shown in the following example:

```
Cursor position recognition  Optional
                             Row: 1   Column: 1

String recognition           Non-optional
                             String 1: Welcome
                             Start position: Row: 1   Column: 6
                             End position:   Row: 1   Column: 12

                             String 2: Username
                             Start position: Row 20   Column 10
                             End position:   Row 20   Column 17
```

In this example, HATS must find both text strings or the cursor position for the screen to match. Because HATS checks non-optional criteria first, HATS looks for the text strings first. If HATS cannot find both text strings in the specified regions of the host screen, then it checks to see if the optional criterion (cursor position) can be found.

## Inverted match of screen recognition criteria

You can choose whether the screen recognition criteria you set matches or does not match the host screen. If you check the **Invert** checkbox, the recognition criterion must not match the screen for the criterion to be considered true.

Conversely, if you do not check the **Invert** checkbox, the recognition criterion must match the screen for the criterion to be considered true..

How you use the **Invert** checkbox corresponds to the Host On-Demand screen descriptor attribute, **invertmatch**.

# Actions tab

During project creation or customization, when you specified screen recognition criteria that HATS uses to match host screens, you also defined the actions to be taken when the host screen is recognized. The **Actions** tab of the screen customization editor displays the actions that you defined for the screen

customization. These actions are applied in the order that they are listed. If you want to change the order of the actions, select one and click **Up** or **Down** to move that actions higher or lower in the list.

You can add, edit, or remove actions on this tab. Choose from the following actions:

- **Apply transformation**
- **Insert global variable** (or a text string onto the host screen)
- **Extract global variable**
- **Set global variable**
- **Execute business logic**
- **Show a URL**
- **Play a macro**.

To play a macro, click the checkbox and select the name of the macro to play from the drop-down list. If you define a macro to be played as an action of this screen customization, it is the last action applied. You can record macros in the HATS Studio using the host terminal. You can also import macros created with other programs, such as the IBM WebSphere Host Publisher Host Access application or IBM Host On-Demand MacroManager. For more information on importing macros, see Chapter 8, "Incorporating macros" on page 37.

All other action types and their descriptions are shown in the table on the **Actions** tab. If you highlight a row of the table and click **Edit**, the **Edit xxx Action** dialog appears. where **xxx** is one of the following:

- Apply
- Insert
- Extract
- Set
- Execute
- Show.

If you click **Add**, the **Add Action** dialog appears. The dialog panel shows the screen selected on the **Overview** tab. In this panel, you can select the action you want to occur from the drop-down list. Depending on the action you choose, the rest of the panel displays information that you can specify for that action.

**Note:** You cannot change the action in the **Edit xxx Action** dialog. You can only change the information that applies to the action.

## Apply transformation action

If you decide to apply a transformation as the action of this screen customization, you can select the transformation you want to apply from the drop-down list of transformations defined in the project.

The **Template** field has "**(default template)**" selected by default. Unless you select a different template to be applied with this particular transformation, the template that surrounds the transformation in the browser window is the template you specified as the default template for the project. The drop-down list contains all the templates defined in the project.

If you want host keys pressed by the end user of your project to be sent to the host immediately instead of waiting until all actions have been performed, click

**Immediate host keys**. On this panel, click the checkboxes of the keys that should be sent to the host immediately. The immediate sending of these keys applies only to the current transformation, and not for all transformations in the project.

## Insert global variable action

You can insert information onto the host screen. Select whether the information is a string or a global variable by clicking the appropriate radio button. To insert a string, type the text in the entry field provided. To insert a global variable, select the name of an existing global variable from the drop-down list. In either case, specify the row and column of the screen where the variable should be inserted.

**Note:** Inserting information onto a host screen must occur **before** any transformation occurs for the global variable to appear in the Web page. See "Actions tab" on page 19 for information on modifying the order of the actions.

If the value of the global variable is indexed (contains a list of strings), click **Advanced**. You must select one of the radio buttons to specify whether all of the strings are inserted at the specified position one after the other or if the strings are inserted as separate lines into a rectangular region of the screen.

For more information about global variables, see Chapter 7, "Interacting with global variables" on page 35.

## Extract global variable action

You can extract information from the screen and define it as a global variable. When you extract a global variable, you can specify a name or select an existing global variable name from the drop-down list for the **Name** field. For the region of the host screen, you define the starting and ending rows and columns for the area of the screen you want to assign as a global variable.

To specify how text extracted from multiple rows of the host screen is defined in a global variable, click **Advanced**. You must select one of the radio buttons to specify if the extraction should be treated as one string or as a list of strings (indexed). If you selected an existing global variable in the **Name** field before you clicked **Advanced**, you must select one of the radio buttons specifying how HATS should handle the extracted data.

There are four options for handling extracted data for existing variables:

- Overwrite the existing value with this new value
- Overwrite the existing value with this new value, starting at the specified index
- Append this new value to the end of the existing value
- Insert this new value into the existing value, at the specified index.

For either of the options that use a specified index, you must enter the number of the index in the **Index** field.

The following example illustrates how the variable value is modified based on the option you choose. Start with an existing indexed variable named "sample". The values of "sample" are "a b c d". The "a" in the value has an index of 0, so the value of "sample[0]" is "a", and the "b" in the value has an index of 1, so the value of "sample[1]" is "b", and so on. Assume that you extract a new set of values "e f g".

- If you select the **Overwrite the existing value with this new value** radio button, the value “a b c d” of “sample” is changed to “e f g”.
- If you select the **Overwrite the existing value with this new value, starting at the specified index** radio button, and assume an index of 2, the value “a b c d” of “sample” becomes “a b e f g” .
- If you select the **Append this new value to the end of the existing value** radio button, the value “a b c d” of “sample” becomes “a b c d e f g”.
- If you select the **Insert this new value into the existing value, at the specified index** radio button, and assume an index of 2, the value “a b c d” of “sample” becomes “a b e f g c d”.

For more information about global variables, see Chapter 7, “Interacting with global variables” on page 35.

## Set global variable action

You can set global variables to be used by other objects within your project. When you set a global variable, you can specify a name or select an existing global variable name from the drop-down list for the **Name** field. If you select an existing indexed global variable, click **Advanced** to specify how to handle the setting of the value.

There are four options for setting the value for an existing indexed variable:

- Overwrite the existing value with this new value
- Overwrite the existing value with this new value, starting at the specified index
- Append this new value to the end of the existing value
- Insert this new value into the existing value, at the specified index.

For either of the options that use a specified index, you must enter the number of the index in the **Index** field.

For an example of how variable value is set based on the option you choose, see “Extract global variable action” on page 21.

You can specify whether the global variable is set to a fixed constant or a calculated value by clicking one of the radio buttons. If you are setting the global variable to a fixed value, type the value in the entry field.

If you are setting the global variable to a calculated value, you specify the operands to be used and the operation of the calculation. The operands can be either fixed values that you enter into the field, or you can use the values of existing global variables for the calculation. If you use an existing indexed global value, and you want to specify an index of the variable to use as the operand, click **Advanced**. Enter the number of the index in the **Index** field.

For more information about global variables, see Chapter 7, “Interacting with global variables” on page 35.

## Execute business logic action

If you decide to execute some business logic as the action of this screen customization, you must specify the fully-qualified Java class name and the Java method for the business logic you want to perform in the fields provided. You can click **Browse** next to the **Class name** field to select a class in which the business logic method is defined. You can select any class defined under the `Source` folder in the **HATS Project View** tab of the HATS Studio. If you have not created the

Java code for this business logic, right-click in the **HATS Project View** tab of the HATS Studio, and select **New HATS > Business Logic** to invoke the Create Business Logic wizard.

For more information about business logic in your projects, see Chapter 9, "Adding business logic" on page 41.

### Show URL action

If you want to show a Web page as the action of this screen customization, you must specify the URL (uniform resource locator) address of the Web page in the **URL** field. With Internet Explorer version 5.0 or higher and Netscape version 6.0 or higher, the Web page will be shown surrounded by the default template, similar to the way a transformation is shown. With Netscape version 4, the URL is a link surrounded by the default template. When the end user clicks the link, a new window opens with the specified URL.

## Source tab

The **Source** tab displays the tags and values in the *sc-name*.evnt file for all the information supplied for the screen customization, where *sc-name* is the name you gave to the screen customization when you created it. As you make changes on other tabs in the screen customization editor, the tags and values displayed in the source file change to match.

You can also make changes to the tags and values in the source file, and they are reflected on the appropriate tabs of the screen customization editor. For information about the tags in the *sc-name*.evnt file, refer to "Screen customization files (.evnt)" on page 115.

## Screen customization ordering

HATS checks each screen customization in the order that you arranged them during project creation. You can modify the order of the screen customizations using the **Event Priority** tab of the project editor. See "Event Priority tab" on page 13 for more information.

When a HATS application is running and a new host screen is found, the first enabled screen customization in the event priority list is checked to determine if the screen recognition criteria match the host screen. If so, no more screen customizations are checked for matches, and the actions for the first screen customization are performed. If not, the next screen customization in the list is checked to determine if the screen recognition criteria match the host screen. This continues until the last screen customization in the list is checked.

If there are no screen recognition criteria in the screen customizations that match the current host screen, HATS processes the unmatched screen event. The HATS unmatched screen event is a special screen customization that occurs only when no defined screen customizations match the host screen. The default action of this event is to display the host screen (default transformation) applying the default template.

You can modify the actions to be taken if a host screen that does not match any of your screen customizations. For example, you could create a Web page that tells the end user that the page was not found and gathers information on how the user reached that screen. You could use the show URL action to present the Web page.

If you want to modify the action of the unmatched screen event, you can locate the `unmatchedScreen.evnt` file in the `source/profiles/events/session/main` directory path of the **Navigator** tab of the HATS Studio. You can invoke the screen customization editor by double-clicking on the `unmatchedScreen.evnt`. Click the **Actions** tab at the bottom of the editor to display the actions. Click **Add**, **Edit**, or **Remove** to customize the `unmatchedScreen.evnt` file.

# Chapter 5. Editing a transformation

A transformation is a JSP file that defines how to customize specific host screens of your HATS project. Applying a transformation is one of the possible actions of a screen customization. Some common uses of transformations are:

- Rearranging the presentation of host screen information
- Filtering host screen information that you do not want to show to end users
- Presenting host components as widgets in the Web presentation.

You use the Create a HATS Transformation wizard to define a transformation. The name, description, and the screen you define for the transformation are saved in a transformation (.jsp) file. You can use the HTML editor built into WebSphere Studio to view and modify the information you define for the transformation. See the WebSphere Studio documentation for more information on using the HTML editor features.

You can see the transformations you have created by expanding the **Transformations** node of the **HATS Project View** tab of the HATS Studio. To edit a transformation, you use the HTML editor built into WebSphere Studio. You can invoke the HTML editor by double-clicking on the name of the transformation.

The following restrictions apply to creating transformations:

- Transformations must be UTF-8 encoded.
- Do not use any JSP variable, CSS class, HATSForm, or any other object that starts with HATS, hats, or Hats. These names are reserved for use by HATS.

The following sections describe each tab of the HTML editor.

## Design tab

The **Design** tab displays the current view of the transformation as you make changes to it. While on this tab, you can insert text, graphics, global variables, host components, tabbed folders, macro keys, host and application keypads, or individual keys from the keypads.

**Note:** HATS automatically inserts the default host keypad into each transformation you create.

You can select these items using the pull-down menus on the menu bar, or the **Insert HATS Component** drop-down menu on the HATS Studio toolbar.

If you want to add images to your project, such as those in the WebSphere Studio **Gallery**, it is recommended that you import them into the `Common/Images` directory of your project. To import images, select **File > Import > File System** to open the Import wizard. Select the location of the image source files you want to import in the **Directory** field. Select the *project_name*`/webApplication/common/images` directory as the destination **Folder**. When your image source files are imported, right-click the on the `Images` folder, and select **Show thumbnails** to see the images in the folder on the **Thumbnail** tab in the lower right window. You can use the drag-and-drop method to copy images into the **Design** tab view of your transformation.

You can also import images from the WebSphere Studio **Gallery** tab.

When you click the **Insert Host Component**, **Insert Tabbed Folder**, **Insert Macro Key**, or **Insert Global Variable** items on the **Insert HATS Tags** menu, a wizard appears for you to define those items.

## Insert Host Component wizard

With the Insert Host Component wizard, you select the screen from which you want to extract a host component. You also select a region on the screen from which to extract a host component by drawing a rectangle around the text. Place your cursor at any point on the screen, click and hold the left mouse button, and move the cursor to another location on the screen to draw the rectangle. The fields at the bottom of the wizard show the starting and ending row and column numbers of the rectangle. You can also enter the row and column numbers by typing the numbers in the fields. If you want to see where the input fields are defined on the screen, click the **Highlight input fields** checkbox. When you have selected the starting and ending row and column numbers of the screen, click **Next** to display the rendering options for the host components found in the selected region.

HATS provides host components and widgets. You can choose one of the host components and widgets provided or you can create your own custom host components and widgets. For information on creating custom host components and widgets, see "Creating custom host components and widgets" on page 55.

Click one of the components in the **Component List**. The Component Preview window displays the component if it is found in the screen region. You select the widget to use to render the host component from the widgets in the corresponding **Widget List** that are available to render the component. When you select a widget, the Widget Preview window displays how the widget is displayed in the final Web page. A larger widget preview is available if you click the **Widget preview in large window** (the magnifying glass). You can click **Full page preview** to show all the components on the page along with the associated template. This preview shows the page as it will appear to the end user.

The default project settings for components and widgets are configured using the **General** tab of the project editor. Some components and widgets have settings that you can customize by clicking **Component Settings** or **Widget Settings**. You can also customize the component and widget settings for a particular transformation using the Insert Host Component wizard, by clicking **Component Settings** or **Widget Settings**. For information on the settings that can be customized with the Insert Host Component wizard, see "Component and widget settings" on page 89.

The widgets that are available depend on the selected host components. Table 4 on page 101 lists the existing HATS host components and their corresponding widgets.

If HATS does not find the component in the screen region, the Component Preview window displays the message "The selected region does not contain the *component_name* component", where *component_name* is the component selected in the **Component List**. If this message is displayed, you might not have selected a region that contains the complete component, or you might need to modify the settings of the component to match the way your host application displays the component. For example, you may have a Command Line component in the

region, but your command line uses the token >>> instead of ==>, so you could change the token attribute of the Command Line component to look for a command line with the correct token.

Refer to Appendix A, "Component and widget descriptions and settings" on page 89 for a list of the host components and widgets.

Click **Finish** when you have made your component and widget selections.

## Insert Tabbed Folder wizard

Use this wizard to insert a folder with tabs into your Web page. Tabbed folders are helpful in organizing your components and information when you have a large amount of information to display on the Web page. With the Insert Tabbed Folder wizard, you specify how many tabs you want for your folder. For each tab in the folder, you also specify the following:

- The label text for the tab
- The host component you want to display on the tab
- The background color for the tab when it is selected
- The background color for the tab when it is not selected

Under the **Tab advanced options**, if you clear the **Use default values** checkbox, you can specify the following:

- The color of the text on the tab when the tab is selected
- The color of the text on the tab when it is not selected
- The color of the tab when you place your cursor over the tab

Under the **Folder advanced options**, if you clear the **Use default values** checkbox, you can specify the following:

- The height of the tab in pixels
- The width of the folder in pixels
- The height of the folder in pixels
- The color of the folder outline

The Preview window shows how the tabbed folder will appear, based on the selections you make.

Click **OK** when you have defined all of the tabbed folder options for each tab in the folder.

You can click **Full page preview** to show all the components on the page along with the associated template. This preview shows the page as it will appear to the end user.

## Insert Macro Key wizard

With the Insert Macro Key wizard, you can display a macro on the Web page presented to the end user. The end user could execute the macro by clicking on a button or a link, or by selecting the macro from a drop-down list. For example, your Web page could present a logon screen to the end user, which also has a button for a logon macro. When the end user clicks the button, the macro plays to supply a user ID and a password, and navigates to the next screen that the end user needs to see.

To add a macro to your Web page, select one or more macros to add from the list of macros defined in the project by clicking the checkbox next to the name of the macro. You also define how to display and initiate the macro from the Web page. Choose from one of the following:

- Individual button
- Individual link
- An item in a drop-down list.

**Note:** HATS uses the description of the macro as the text inserted into the Web page for any of the rendering options. You might want to consider this when providing a description of the macro, and avoid giving macros long descriptions.

### Insert Global Variable wizard

With the Insert Global Variable wizard, you select a defined global variable from the drop-down list for which you want to display the value. If the value of the global variable is indexed (contains a list of strings), click **Advanced** to display the **Handle Indexed Variables** dialog. If you click the **Variable is indexed** checkbox, you can select one of the radio buttons to specify whether all indices or only a single index is inserted. If you select the **Show a single index** radio button, you also specify the number of the index to be inserted.

## Source tab

The **Source** tab displays the HTML and JSP tags in the *transformation*.jsp file necessary for extracting host components from the host screen, the widgets you selected to present those host components, and any other items you added to the transformation. As you make changes on other tabs in the HTML editor, the tags and attributes displayed in the tags of the source file change to match.

You can also make changes directly to the tags and attributes in the source file, or you can insert items using the **Insert HATS Component** drop-down menu on the HATS Studio toolbar. The items you can insert on the **Source** tab are the same items listed on the **Design** tab. Place your cursor in the source file at the point you want to insert one of the menu items.

When a host component and its rendering widget have been inserted, you can use the **Edit HATS Component** toolbar option to modify the host component and widget. Before you click **Edit HATS Component**, make sure your cursor is inside of the <HATS:Component> tag.

When you make changes to the file displayed on the **Source** tab, they are reflected on the appropriate tabs of the HTML editor.

## Preview tab

The **Preview** tab provides a browser preview of the transformation showing the static HTML content. This is similar to the **Design** tab, but without the ability to make changes to it. Items defined with the <HATS:Component> tag are not shown in the preview. There are other ways to preview your transformation, along with its associated template.

As mentioned on the **Design** tab, you can click **Full page preview** in the Insert Host Component wizard or the Insert Tabbed Folder wizard to show all the components on the page along with the associated template. This preview shows the page as it will appear to the end user.

Another way to preview your transformation is to use captured screens. For every transformation you create in your project, HATS creates a captured screen (whether you requested it or not). You can see the captured screens by expanding the **Screen Captures** node of the **HATS Project View** tab of the HATS Studio. Double-clicking on the name of the screen capture displays a view of the screen capture with two tabs, **Host Screen** and **Preview**. The **Host Screen** tab displays the captured screen as it appears on the host. The **Preview** tab displays how the transformation is rendered on the Web page, along with the template associated with the transformation.

The transformation and template used to generate the preview are based on screen customizations defined in your project. **Preview** scans the list of enabled screen customizations. When a screen customization is encountered that matches the captured screen, the first action that applies a transformation (along with the associated template) is used to render the preview. If no matching screen customization is found, the default template and transformation are used for the preview.

# Chapter 6. Using templates

Templates enable you to further control the appearance of your HATS project. A template is a JSP file with an area reserved for the host screen that is rendered by a HATS transformation. The template can contain company logos and information and links to other Web pages. A template also defines the background color or image (or both) for the area where the transformed host screen appears.

HATS supplies templates that you can use in your projects. You can see the names of these templates by expanding the **Templates** node of the **HATS Project View** tab of the HATS Studio. The supplied templates contain HTML and JSP code to include some or all of the following:

- At least one stylesheet (.css) file
- Borders created using the .gif and .jpg files (located in the `Common/Images` node)
- The HATS default application keypad
- An area for the host screen rendered with a transformation.

When you created your project in HATS Studio, you selected a template to use as the default template for your project.

When you create screen customizations, and the action is to apply a transformation, you can select the template to use when the transformation is applied. To make your host application consistent across all screens, you can use the same template for your transformation that you selected for your project default template. You do this by allowing the template to default to the template selected for the project, selecting **(default template)** from the drop-down list. The **(default template)** selection is the default when applying a transformation is the action of a screen customization.

## Creating your own templates

You can create your own custom templates to meet functional needs and corporate style guidelines. You can design custom templates for your projects using the wizards and editors in HATS Studio. HATS automatically adds the necessary code to include a stylesheet, the HATS default application keypad, and an area for the host screen rendered with a transformation to any templates that you create. When you create your own template, ensure that the following required tags are included:

**<LINK rel="stylesheet" href="xxx"> tag**
One or more tags that refers to a stylesheet to define each of the following:
- Appearance of the buttons, links, and keypads
- Colors for the template background and widgets
- The font and size for text.

**<HATS:Transform>**
This tag defines the location of the transformation that the template surrounds.

You can see all of the templates, those supplied by HATS and any that you have created, by expanding the **Templates** node of the **HATS Project View** tab of the HATS Studio. To edit a template, you use the HTML editor built into WebSphere

Studio. You can invoke the HTML editor by double-clicking on the name of the template. See the WebSphere Studio documentation for more information on using the HTML editor features.

There are two ways to create your own template:

- Click **Create HATS Template** on the HATS Studio toolbar to start a new template.
- Edit one of the supplied templates and make changes to it.

If you want to use your new template as the default template for your project, make sure you select the name of your template as the default for your project in the project editor.

The following restrictions apply to creating custom templates:

- Custom templates must be UTF-8 encoded.
- Do not use any JSP variable or CSS class that starts with HATS. These names are reserved for use by HATS.
- Do not create a form named HATS_form. HATS generates this form while assembling HTML outputs.
- Do not use a form that encloses the HATS:Transform tag. HATS generates a form in the place where the HATS:Transform tag is. HTML does not allow nested forms.

Keep in mind that any changes you make to objects in a project only affect that project. If you want to use the template you create for other projects, you need to copy that template to any new projects you create.

The following sections describe each tab of the HTML editor.

## Design tab

The **Design** tab displays the current view of the template as you make changes to it. While on this tab, additional edit options are available from the WebSphere Studio toolbar. For example, you can use the **Insert** drop-down menu from the toolbar to insert things such as images, photographs, text, and text-formatting controls. You can add music that plays when the template is displayed, create layout frames, and add HTML tags. You can also insert global variables, macro keys, host and application keypads, or individual keys from the keypads.

You can select these items using the pull-down menus on the menu bar, or the **Insert HATS Component** drop-down menu on the HATS Studio toolbar.

If you want to add images to your project, it is recommended that you import them into the `Common/Images` directory of your project. To import images, select **File > Import > File System** to open the Import wizard. Select the location of the image source files you want to import in the **Directory** field. Select the *`project_name`*`/webApplication/common/images` directory as the destination **Folder**. When your image source files are imported, right-click on the `Images` folder, and select **Show thumbnails** to see the images in the folder on the **Thumbnail** tab in the lower right window. You can use the drag-and-drop method to copy images into the **Design** tab view of your template.

When you click the **Insert Macro Key** or **Insert Global Variable** items on the **Insert HATS Tags** menu, a wizard appears for you to define those items, as you

can in transformations. See “Insert Macro Key wizard” on page 27 and “Insert Global Variable wizard” on page 28 for more information about these wizards.

### Using stylesheets

You can control elements of output such as font color, size, and background color in order to maintain the consistency of the area of the screen rendered by HATS with the style of the template. For example, Cascading Style Sheets (CSS) is a simple style language that enables attaching style to HTML elements.

HATS provides stylesheets to modify color schemes and font size. At least one of these stylesheets is applied to the template. While viewing the template on the **Design** tab, you can apply these stylesheets to your template. Right-click on the Stylesheets folder and select **Show thumbnails** from the pop-up menu. The stylesheet files are shown in the **Thumbnails** view below the **Design** tab. To apply one of the stylesheets to your template, double-click the stylesheet.

To change the output style of HATS templates, you can edit a stylesheet that was shipped with HATS. The stylesheets that HATS provides are located in the `Common/Stylesheets` node of the **HATS Project View** tab of the HATS Studio. Double-click on any stylesheet to edit the file. Read the comments in the file to determine the functions of the styles included in the stylesheets.

## Source tab

The **Source** tab displays the HTML and JSP tags in the *`template`*`.jsp` file for all the parts of the template. As you make changes on other tabs in the HTML editor, the tags and attributes displayed in the tags of the source file change to match.

You can also make changes to the tags and attributes in the source file, and they are reflected on the appropriate tabs of the HTML editor.

## Preview tab

The **Preview** tab provides a browser preview of the transformation showing the static HTML content. This is similar to the **Design** tab, but without the ability to make changes to it. This preview does not include the transformation.

# Chapter 7. Interacting with global variables

A global variable contains a value that can be used to pass information from one HATS object to another. For example, you can extract information from several locations on a host screen, perform calculations, and insert the result on the current screen or a future one. You can build up an array of strings from one or more host screens and insert them into a transformation. You can extract a string that a user enters into a field on a Web page and use it elsewhere.

Global variables exist for the time that the HATS application is active. If several users open host sessions using the same HATS application, the global variables for each session are used only in that session; they are not shared between different sessions using the same HATS application. A global variable can contain a numeric value, a string, or an indexed array of strings. If you use a global variable to contain an array of strings, you can specify for any action whether you want to use the entire array, a particular index, or all the values starting at a particular index. All operations on global variables are case-sensitive. Do not use names beginning with "HATS" for global variables.

You can set the value of a global variable in these ways:

- With a **Set global variable** or **Extract global variable** as an action on a screen customization or other event
- By prompting the user for a value while running a macro
- By setting the value in a business logic program.

After a global variable has a value, you can use that value in the following ways:

- To calculate the value of another global variable, in a **Set global variable** action
- To write the value to a host screen, using an **Insert global variable** action
- To insert the value into a transformation or a template, using the **Insert global variable** menu item
- To pass the value to a macro
- To use the value in business logic.

If you insert a global variable into a host screen, you must list this action before applying a transformation, so that the global variable will appear on the Web page created from the host screen. See "Actions tab" on page 19 for more information about specifying actions for screen customizations. For information on inserting global variables into transformations and templates, refer to "Insert Global Variable wizard" on page 28.

Global variables can be used with prompt and extract macros to either provide a value for a prompt or to store a value extracted from the host screen. See Chapter 8, "Incorporating macros" on page 37 for more information about using global variables with macros.

To use global variables in business logic, you must check the **Get global variable** box in the Create Business Logic wizard. This creates a stub in your business logic code to give you access to HATS global variables. See Chapter 9, "Adding business logic" on page 41 for more information about using business logic.

If you want to use a global variable to accumulate strings or a numeric value from several screens, you can initialize it by adding a **Set global variable** action to the connect event. From the **HATS Project View** tab of the HATS Studio, click the **Navigator** tab at the bottom, expand your project, and expand the `source/profiles/events/session/main` directory. Double-click **connect.evnt** to customize the connect event.

# Chapter 8. Incorporating macros

HATS supports macro-based customization to speed up the host application process. HATS supports skip-screen macros, prompt macros, and string extract macros. Skip-screen macros are navigational macros that move the user from one screen to another screen without displaying intervening screens. When running a skip-screen macro, HATS does not display any of the bypassed screens of the macro to the user. After the skip-screen macro has finished, HATS processes the ending screen (comparing the screen to the defined screen recognition criteria).

Prompt macros contain events to request input from users during the host session. For example, you can use a prompt macro to ask a user for their user ID and password before logging them into a host application.

Extract macros contain events to extract host screen information as a string. You can use an extract macro to connect to a directory-type host application and extract the results of doing a search in the directory. For example, you can use an extract macro to extract the results of a search for ″Smith″ in a phone book application.

You can record macros in HATS Studio using the HATS host terminal. On the HATS host terminal screen, click **Record Macro**. The Record a Macro wizard appears and enables you to name the macro, give it a description, and specify where the macro is saved. Click **Finish** when you have specifed these items. You can then use the HATS host terminal screen to navigate through the host application to any screen.

If you want the macro to prompt the user for information, click **Insert Prompt** to display the Insert Prompt wizard. You can give the prompt a name and a default value. If the information the end user provides, such as a password, should not be displayed on the host screen, click the **Password protect input** checkbox. The **Row** and **Column** fields of the **Position** section of the wizard define where on the host screen the prompt information provided by the end user is placed. If you place your cursor at a location on the host screen, such as the field for a password, before you begin recording the macro, the **Row** and **Column** fields are filled with those values. The **Handle Macro Prompt** section of the wizard enables you to determine how the prompt is processed. You can select one of the following radio buttons:

**Show 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 Studio 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.

**Set prompt to string**
: If you know what value should be returned from a prompt, you can enter that string in the **String** field.

**Set prompt to global variable**
: If you want the value of the prompt to be provided by a global variable,

enter a name for the global variable in the **Name** field or select an existing variable using the drop-down menu next to the **Global variable** field.

Click **OK** when you have made your selections.

If you want the macro to extract information from the host screen, click **Insert Extract** to display the Insert Extract wizard. You can specify a name for the extract. The **Start row**, **Start column**, **End row**, and **End column** fields of the **Position** section of the wizard define from where on the host screen the information is extracted. If you mark a region of the host screen with a rectangle after you click **Insert Extract**, the **Position** section fields are filled with the values when the Insert Extract wizard is displayed. The **Handle Macro Extract** section of the wizard enables you to determine how the prompt is processed. You can select the following check boxes:

**Show 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 Studio 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.

**Save as global variable**
You can enter a name for the global variable in the **Name** field or select an existing variable using the drop-down menu. You must specify the extraction format by selecting one of the following radio buttons:

- Extract this region as one string
- Extract this region as a list of strings.

If you selected an existing global variable in the **Name** field, you must specify how to handle the existing variable by selecting one of the following radio buttons:

- Overwrite the existing value with this new value
- Append this new value to the end of the existing value.

Click **OK** when you have made your selections.

You can also import macros recorded with other programs, such as the IBM WebSphere Host Publisher or IBM Host On-Demand. To import these macros, select **File > Import > HOD/Host Publisher Macro** and click **Next** to display the **Import a HOD/Host Publisher Macro** dialog. Click **Add** and navigate to the location of the macro on the file system. Host Publisher macros are typically in a directory path `\hostpub\Studio\IntegrationObjects`.

**Note:** Host Publisher macros with fixed iteration loops continually recognize the same screen and perform different actions. In HATS, you cannot create a screen customization to recognize the same screen a second time and perform a different action than the first time it was recognized. If you attempt to use a Host Publisher macro with fixed iteration looping, your project might go into an infinite loop. Host Publisher macros with fixed iteration looping can be identified by looking at the source code for the macro. The macro contains `customreco` and `custom` tags with ID attributes of `HPubFixedIterationLoop` and `HPubIncrementLoop`, respectively.

When a macro is listed in a project, it can be defined to run as the action of a screen customization, when the screen recognition criteria match the host screen. Playing a macro must be the last action performed for a screen customization.

You can also create a button to play the macro using the transformation editor. See "Insert Macro Key wizard" on page 27 for more information.

Macros recorded or imported in HATS Studio are saved in a HATS macro (.hma) file. You can use the macro editor to view and modify those macros.

You can see the macros defined in your project by expanding the **Macros** node of the **HATS Project View** tab of the HATS Studio. You can invoke the macro editor by double-clicking on the name of the macro.

The following sections describe each tab of the macro editor.

## Overview tab

The **Overview** tab of the macro editor summarizes information about the macro, such as the name and description. The only item you can modify on this tab is the description of the macro.

You can click **Editor** to launch the Host On-Demand macro editor, and modify settings for the macro. A separate window opens for the macro editor. Refer to Host On-Demand help documentation for more information on the Host On-Demand macro editor.

## Prompts and Extracts tab

The **Prompts and Extracts** tab of the macro editor lists the configured macro prompts and extracts, and how they are handled when the macro is played. You can edit the HATS-specific properties of a prompt or an extract by selecting it in the table and clicking **Edit**.

## Source tab

The **Source** tab displays the tags in the *macro-name*.hma file for all the attributes and values for the macro, where *macro-name* is the name you gave to the macro when you created it or imported it. As you make changes on other tabs in the project editor, the tags and attributes displayed in the tags of the source file change to match.

You can also make changes to the tags and attributes in the source file, and they are reflected on the appropriate tabs of the macro editor. For more information about the tags in the *macro-name*.hma file, refer to "Macro files (.hma)" on page 120 and Appendix C, "Macro script syntax" on page 123.

# Chapter 9. Adding business logic

Business logic is any Java code invoked as an action in an event, such as a screen customization. Business logic is specific to the application and is not provided as part of HATS.

You can add business logic to your project using the Create Business Logic wizard. To invoke this wizard, right-click in the **HATS Project View** tab of the HATS Studio, and select **New HATS > Business Logic**. You can also right-click in the **Navigator** tab of the HATS Studio, and select **New > Other > Host Access Transformation Server > HATS Business Logic**, and then click **Next**.

In the Create Business Logic wizard, specify the project to which you want to add the business logic and supply the fully-qualified Java class name. Optionally, you can supply a package name, or select an existing Java package by clicking **Browse**. If you want your business logic to have access to the project global variables, check the **Get global variable** checkbox. Click **Finish** when you have provided the required information.

You can see the business logic files in the project by expanding the Source folder on the **HATS Project View** tab of the HATS Studio. Each package name or class name appears in the Source folder. Expand the package name folder to see the Java class name. Double-click on the class name to edit the class.

If you use the Create Business Logic wizard to create business logic, the method is named "execute" by default. If you write your own class, the method must meet specific requirements:

- Marked public and static
- Have a return type of void
- Take a com.ibm.hats.common.BusinessLogicInfo object as the only parameter

The method must follow the form:

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

followed by your own business logic code.

The BusinessLogicInfo object 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 `javax.servlet.http.HttpServletRequest` class
- The `javax.servlet.http.HttpServletResponse` class
- The connection hashtable, which contains the settings for the connection information you provided for the application
- Class properties, which provide default settings for objects such as components and widgets
- The com.ibm.hats.common.GlobalVariable objects in the application
- The com.ibm.hats.common.HostScreen object, which contains "greenscreen" information
- The `java.util.Locale` class of the client
- The com.ibm.hats.common.TextReplacementList values and settings

- The client session identifier string
- The current screen orientation of bi-directional sessions
- The existence of the **Screen Reverse** button in the browser for bi-directional sessions.

For more information about the classes made available to you, see the Java documentation at the product Web site (http://www.ibm.com/software/webservers/hats) for the `BusinessLogicInfo` 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. Select **File > Import > File System** to open the Import wizard. In the Import wizard, select the location of your source files in the **Directory** field. Select the Source folder of your project in the destination **Folder** entry field. When your source .java files are imported, they are automatically compiled and packaged into your HATS project. You can also edit, set breakpoints, and debug your source files in the WebSphere Studio workbench.

You can also incorporate a Java archive (.jar) file with compiled Java business logic. This method imports the Java archive file into the .ear project. There will only be a single copy of the Java file in the .ear file, but it is available to all of the HATS projects contained in that .ear project. There are three steps to this method.

1. Import the .jar file into the HATS .ear project. Select **File > Import > File System** to open the Import wizard. Select the Java archive (.jar) you want to import in the **Directory** field. Select your HATS .ear project as the destination **Folder**. When your .jar file is imported, click the **Navigator** tab of the HATS Studio and expand your HATS ear project. You will see the imported java archive file.
2. In the **Navigator** tab of the HATS Studio, select the project in which you want to invoke your business logic. Right-click on the high level HATS project and select **Properties**. In the **Properties** dialog, select **Java Build Path** in the left table and select the **Libraries** tab on the right. Click **Add JARs** to display the **JAR Selection** dialog. Expand the HATS .ear project, and select the newly import Java archive file. Click **OK** in the **JAR Selection** dialog, and click **OK** in the **Properties** dialog. Repeat this process for all HATS projects for which you want to use the business logic.
3. In the **Navigator** tab of the HATS Studio, select the project in which you want to invoke your business logic. Expand the project, the webApplication folder, and the META-INF folder. Double-click on the MANIFEST.MF file. Type in the name of your newly imported jar at the end of the `Class-Path:` line.

There are other ways to import Java archives into the HATS project. HATS projects are extensions of Web projects in the WebSphere Studio workbench. For more information about importing Web projects, open the Help perspective in the WebSphere Studio workbench and select **Application Developer Documentation**. Expand the sections as follows to find information on Web projects: **Concepts > Projects > Web projects**.

# Chapter 10. Integration of Host Publisher objects

Host Publisher and HATS are complementary products you can use together to accomplish a variety of tasks. You can use Host Publisher objects to incorporate data from other legacy systems or databases with data from HATS legacy systems. You can use HATS global variables as a mechanism to share data between HATS variables and Host Publisher inputs and outputs. From HATS business logic, templates, and transformations, you can invoke Remote Integration Objects, EJB Access beans, and Web Services. You can also import Host Publisher macros into your HATS project. See Chapter 8, "Incorporating macros" on page 37 for information on importing macros into HATS.

## Invoking Host Publisher Remote Integration Objects from HATS

You can invoke Host Publisher Remote Integration Objects from HATS. Although Host Publisher Server must be installed somewhere in your network to invoke Remote Integration Objects, it does not need to be installed on the same WebSphere Application Server as HATS.

To invoke Host Publisher Remote Integration Objects into your HATS project, you must first import the Remote Integration Objects into your HATS project. Refer to the section titled "Remote Integration Object Files" in the *IBM WebSphere Host Publisher Programmer's Guide and Reference* for information on importing Host Publisher Remote Integration Objects into WebSphere Studio workbench. After the Host Publisher Remote Integration Objects are imported into HATS, you can call the Remote Integration Objects from your HATS business logic or from HATS templates or transformations, by using in-line Java code enclosed in <%...%>. See the section titled "Programming with Remote Integration Objects" in the *IBM WebSphere Host Publisher Programmer's Guide and Reference* for more information.

To invoke a Host Publisher Remote Integration Object from HATS business logic, right-click in the **HATS Project View** tab of the HATS Studio and select **New HATS > Business Logic**. Enter a Java class name and package name and click **Finish**. A skeleton HATS business logic template is displayed. For a sample of the template updated to execute a Host Publisher Remote Integration Object, see the product Web site (http://www.ibm.com/software/webservers/hats).

## Invoking Host Publisher EJB Access Beans and Web Services from HATS

You can invoke Host Publisher EJB Access Beans and Host Publisher Web Services from HATS. Although Host Publisher Server must be installed somewhere in your network to invoke EJB Access Beans, it does not need to be installed on the same WebSphere Application Server as HATS.

To invoke Host Publisher EJB Access Beans into your HATS project, you must first import the EJB Access Beans into your HATS project. Refer to the section titled "Using EJB Access Beans with Java Application Clients" in the *IBM WebSphere Host Publisher Programmer's Guide and Reference* for information on importing EJB Access Beans into WebSphere Studio workbench. After the EJB Access Beans are imported into HATS, you can call EJB Access Beans from your HATS business logic or from HATS templates or transformations. See the section titled "Programming with Web

Services Integration Objects and EJB Access Beans" in the *IBM WebSphere Host Publisher Programmer's Guide and Reference* for more information.

# Chapter 11. Enabling print support in projects

As you develop a HATS project, you can establish a print session for the associated host application. When the HATS project is running on a WebSphere server, an end user of the application can also print data or display data that is formatted for printing.

When interacting directly with a host application, an end user activates a physical printer to print data from the application. When interacting with a HATS application, the end user does not activate a physical printer. Rather, he or she generates an Adobe Portable Document Format (PDF) file, which can be displayed in a Web browser. The PDF file can also be printed.

**Note:** If a PDF viewer (Adobe Acrobat Reader) is not installed, the user will be prompted to save the file to disk.

This chapter describes the process for enabling print support in your HATS project and for using print support as you develop the project in HATS Studio and as the application is used by an end user.

## Configuring the host print session on 3270 hosts

Before setting up print support for your HATS project, make sure that you (or the system administrator) have performed the following configuration for the host print session:

- VTAM configuration of a switched major node containing the display and printer logical units (LUs).
- The printer LUs must be associated with the host application in your Telnet server configuration.

Refer to the documentation for your 3270 host software for details on how to perform these steps.

## Defining print support for your project

For print support to be available for your project, you must obtain the host name and the port number for the Telnet server from the system administrator. Enter these values when you define the connection settings for your project.

### For 3270 servers

To define print support when your HATS project interacts with a 3270 server, do the following:

- Click the **Enable print support** checkbox on the **Advanced Connection Settings** tab of the project editor. You can also set values for the following:
  1. Paper size
  2. Page orientation, either portrait or landscape.
  3. The name of the font to be used in the PDF file. The list of fonts from which you can choose depends on the codepage setting for the workstation you are using.

  See “Advanced Connection Settings tab” on page 12 for more information.

- Obtain the name of the printer LU from the system administrator.
- In the box for the optional, advanced connection settings for IBM WebSphere Host On-Demand, click **Add**.
- Select the **LUName** parameter using the drop-down list next to the **Name** field, and enter the name of the printer LU you obtained from the system administrator in the **Value** field.
- Verify with the system administrator that the print subsystem is configured and active.
- If running on a VM system, identify the printer to use for print jobs.

When an end user of the HATS application issues a command to print files, the HATS application sends a print job to the printer LU and the HATS runtime converts the print job to PDF format. Once the PDF is formatted, the end user can click **View Print Jobs** on the application keypad to see a list of queued print jobs.

### For 5250 servers

To define print support when your HATS project interacts with a 5250 server, click the **Enable print support** checkbox in your project settings. Provide a URL for the iSeries for Web Access (IWA) Printer Output window. The default URL is `http://hostname/webaccess/iWASpool`, where *hostname* is the name of the 5250 server. See "Advanced Connection Settings tab" on page 12 for more information.

You do not need to perform any additional configuration. When an end user of the HATS application issues a command to print files, IWA converts the host print jobs into PDF format and facilates the download to the end user. The end user can click **View Print Jobs** on the application keypad to display the IWA Printer Output window. In this window, the end user can select the following print options:

1. **PDF device type**
2. **Paper size**
3. **Destination**

Refer to your IWA documentation for more information on these print options.

## Providing documentation for end users

To facilitate use of your HATS project by end users, we recommend that you provide documentation on how to use the print support in HATS. Your documentation—provided either in the application's graphical user interface or in some other easily displayed form (perhaps a link in your template)—should describe:

- Using the functions in the Printer Output window, as described below
- If a PDF viewer (Adobe) is not installed, the user is prompted to save the file to disk.

  **Note:** You might consider adding a link to your application to where the end user can download a free copy of the Acrobat Reader.
- When the HATS application completes, the printer output window closes automatically.
- Any application-specific information you choose to include.

To use HATS print support, an end user should follow these steps:

1. Start the HATS application.
2. Print the files.

3. Click the **View Print Jobs** button. The Printer Output window displays a list of print jobs, if any exist.

   In the Printer Output window, the end user can click the links for their jobs to either **View** or **Delete** the print job. If the user clicks the view link, the print job is displayed as a PDF file in Acrobat Reader, if it is available. If not, the user is prompted to save the file to disk.

   **Note:** While the print jobs are spooling, the user might see the file names for the print jobs in the printer output window, but **View** and **Delete** are disabled until the conversion to PDF format is complete.

# Chapter 12. Enabling keyboard support in projects

Users frequently interact with host applications using special keys on the physical keyboard, such as **F1**, **Attn**, and **Clear**. There are two different ways in which the end users of your HATS projects can send keystrokes to the host:

- By pressing keys on the physical keyboard. The term *keyboard support*, as used in this chapter, refers to this activity.
- By using a *keypad*—a graphical table of HTML buttons or links that represent keys on the physical keyboard. The end user clicks on the desired key in the keypad to send that host key to the host.

  There are two keypads in HATS Studio that you can add to your project:

  – The host keypad, with keys (such as **F1**, **F2**, and **Clear**) that represent host keys. These keys control functions on the host. HATS transformations, by default, include the default host keypad.

  – The application keypad, with keys (such as **Refresh**, **Default**, and **View Print Jobs**) that represent application-level functions. These keys control functions within the HATS project. HATS templates, by default, include the default application keypad. The application keypad keys include:

    **Reset** Clears all the fields on the browser page of any entries made by the end user.

    **Default**
    Turns off any customization of the host screen by a transform and presents the entire host screen as the default view.

    **Reverse Screen**
    Toggles the screen image from a left-to-right image to a right-to-left image or vice-versa, if the application is running on a host with an Arabic or Hebrew codepage. If an Arabic or Hebrew codepage is not selected in the project connection settings, this button does not appear on the application keypad.

    **Refresh**
    Refreshes the current browser screen and performs the current action again.

    **Disconnect**
    Disconnects from the host session. If this key is clicked, a link appears to let the user reconnect to the host.

    **View Print Jobs**
    Shows a list of print jobs that the end user has created. If print support is not enabled for the project, this button does not appear on the application keypad.

    **Toggle Keyboard**
    Toggles support for using the physical keys on the host keyboard. If keyboard support is not enabled for the project, this button does not appear on the application keypad.

    **Note:** The text of the button seen by the end user depends on the state of the keyboard. The button will read “Keyboard on” when keys are being sent to the browser, and “Keyboard off” when keys are being sent to the host.

Using the settings on the **General** tab of the project editor, you can define whether the project displays keypads, even if the tags for the keypads are included in the templates or transformations. You can define a subset of keys to display in the keypads, and whether the keys should appear as buttons or links. You can also add custom keypads or individual keys to your templates and transformations.

**Notes:**

1. The default keypads are not defined using HTML tags, so they cannot be viewed in the **Design** tab of the HTML editor for a transformation.
2. Custom keypads and individual keys are defined using HTML tags, so they are displayed in the HTML editor.
3. PF keys on host screens that are transformed using the default transformation are displayed as buttons or links. These keys are displayed separately from the default host keypad.

Refer to "General tab" on page 13 for more information about the settings for keyboard support and keypads.

This chapter explains how to define keyboard support in your HATS project and contains tips for documenting keyboard support for your end users.

To use keyboard support in HATS projects, the end user's Web browser must be either Internet Explorer version 5.0 or higher or Netscape version 6.0 or higher. Javascript must be enabled in the Web browser.

## Defining keyboard support

To define keyboard support in a HATS project, you need to enable keyboard support in your project, and define the host keypad and the application keypad keys to include. See "General tab" on page 13 for information about the project settings for keyboard support and the keypads.

### Changing the appearance of the keypads

You can change any of the following items for the keypads:

- Background color for the keypad
- Attributes for buttons (both enabled and disabled):
  - Font family
  - Font size
  - Color
  - Background color
- Attributes for links (both enabled and disabled):
  - Font family
  - Font size
  - Color
  - Background color

To change style of a keypad for a specific HATS project, change the *cascading style sheet* that corresponds to the keypad. In the WebSphere Studio workbench, go to the **HATS Project View** tab of the HATS Studio and expand the project name. Expand `Common > Stylesheets`. The default keypad stylesheet is keypad.css.

To modify the keypad, double-click on the keypad.css stylesheet open the editor.

The cascading style sheet defines seven styles:

**table.HostKeypad**
Host keypad background

**table.ApplicationKeypad**
Application keypad background

**input.HostPFKey**
Host keypad PF buttons

**input.HostButton**
Host keypad buttons

**input.ApplicationButton**
Application keypad buttons

**a.HostKeyLink**
Host keypad links

**a.ApplicationKeyLink**
Application keypad links

## Providing documentation for end users

To facilitate use of your HATS project, we recommend that you document the keyboard settings for your end users—either in the application's graphical user interface or in some other easily displayed form (perhaps a link in your template). Your documentation should describe:

- How to turn keyboard support on and off, using **Toggle Keyboard**
- The HATS keyboard mapping, as shown in Table 1
- How the **Enter** key works with tabbed folders; keyboard support must be off when the user presses the **Enter** key to display the contents of a different tab, after using the **Tab** key to move to a new tab. If keyboard support is not turned off, the **Enter** key goes to the host.
- The required level of Web browser for using keyboard support: either Internet Explorer version 5.0 or higher or Netscape version 6.0 or higher.

*Table 1. Key mapping in HATS*

| Button on button bar | Default physical key mapping |
|---|---|
| F1 – F12 | F1 – F12 |
| F13 – F24 | Shift + F1 – F12 |
| ENTER | Enter |
| CLEAR | Esc |
| SYSREQ | Shift + Esc |
| ATTN | Pause/Break |
| PAGEUP | Page Up |
| PAGEDN | Page Down |
| PA1 | Alt + Delete |
| PA2 | Alt + End |
| PA3 | Alt + Page Down |
| PRINT | Ctrl + P |
| HELP | Ctrl + H |

*Table 1. Key mapping in HATS (continued)*

| **Button on button bar** | **Default physical key mapping** |
|---|---|
| Enable/disable keyboard | Ctrl + K |
| Default | Alt + Insert |
| Refresh | Alt + Page Up |
| Reset | Ctrl + S |
| Disconnect | Ctrl + D |
| View print jobs | Ctrl + J |
| Reverse | Ctrl + R |

# Chapter 13. Enabling SSL security

If you click the **SSL enabled** checkbox in the **Connection Settings** of either the HATS Studio Create a Project wizard or the project editor, you request that data flowing over the connection to be encrypted to secure the connection.

HATS Studio uses Host On-Demand to provide connection support from HATS applications to 3270 and 5250 applications using Telnet protocols. HATS uses the SSL support provided by Host On-Demand for securing these connections. Using a secure connection over SSL encrypts data flowing over the connection and thus protects it against observation by a third party.

For a connection to be secured, both the HATS application and the Telnet server it is connected to must support SSL. To secure the connection, the Telnet server must provide a certificate, which is used to encrypt the data. This certificate uniquely identifies a machine on one end of the connection.

HATS verifies that the certificate is signed by a well-known certificate authority. The well-known certificate authorities for HATS are Thawte, Verisign, and RSA. If the server has a certificate signed by a well-known certificate authority, the certificate is guaranteed to be unique and secure, and HATS is not required to have a copy of the certificate. If the Telnet server has a self-signed certificate, the administrator of the server generated the certificate based on his or her own information without using a certificate authority. In that case, HATS must have a copy of the self-signed certificate to secure the connection.

If a certificate is required by HATS, you can click **Import** in the **Connection Settings** of either the HATS Studio Create a Project wizard or the project editor to import the required certificate from the Telnet server. When the certificate is imported, HATS builds a database called CustomizedCAs.class. The database contains the certificate that you import. The certificate becomes a part of the HATS project, and it is packaged with the rest of the project files when you assemble the project.

# Chapter 14. Creating custom components and widgets

Though basic project customization can be accomplished using the Studio, HATS provides a programming environment for further customization of host applications. You can program your HATS project to define new host components or widgets or modify existing host components or widgets.

For a detailed description of host components and widgets, see "Understanding HATS key concepts and objects" on page 4.

The following sections describe how to create custom host components and widgets.

**Note:** If you are using a bi-directional (BIDI) code page, please refer to "Bi-directional APIs" on page 87.

## Creating custom host components and widgets

HATS creates a JSP page (and writes information to a .jsp file) that reflects host component and widget selections made during transformation configuration in the HATS Studio. 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.

The following JSP code example shows the format of the HATS:Component tag.

```
<HATS:Component type='<HostComponentType>'
   widget='<WidgetStyle>'
   row='1' col='1' erow='24' ecol='80'
   label='<Data on the screen in region (1,1) to (24,80)>'
   componentSettings='' widgetSettings='' />
```

The attribute data of the HATS:Component tag determine what host component and widget classes to call and how to present the widget in HTML output. Refer to "HATS:Component tag type and widget attributes" on page 102 for the values of the type and widget attributes. The rest of the attributes are described in the list that follows:

| Attribute | Description |
|---|---|
| **row** | The starting row position for host component definition. |
| **col** | The starting column position for host component definition. |
| **erow** | The ending row position for host component definition. |
| **ecol** | The ending column position for host component definition. |
| **label** | The string to be rendered as HTML text coupled with this host component. |

**componentSettings**
This is a set of key and value pairs that are 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 an "or bar" ( | ). For example, `<.. componentSettings="key1:value1|key2:value2" ... >`.

The componentSettings attribute can be used for advanced customization of the component. For example, if your command line uses the token ″>>>″ instead of ″==>″, you can pass this component setting value here.

**Note:** The componentSettings and the widgetSettings need to be combined into a single properties object when they are passed to the `recognize()` method of the component, and the `draw()` method of the widget. This allows the component to update any widget settings necessary.

**widgetSettings**

This is a set of key-value pairs that are 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 an “or bar” ( | ). For example, `<.. widgetSettings="key1:value1|key2:value2" ... >`.

The widgetSettings attribute can be used 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.

Following is a description of how HATS processes each HATS:Component tag that it encounters in the JSP page.

1. HATS instantiates a component object based on the **type** attribute setting.

   If HATS recognizes the value of the **type** attribute as one of its components, HATS instantiates the appropriate component. If HATS does not recognize the value of the **type** as one of its components, HATS attempts to instantiate the custom component. You must specify the fully qualified path, like `com.company.division.product.MyComponentExtract`, and place the class file in either the `WEB-INF/classes` directory or in a jar file in the `WEB-INF/lib` directory. For example, if the **type** attribute is set to SelectionList, HATS instantiates the `com.ibm.hats.component.SelectionListExtract` class.

   For a list of valid values for the **type** attribute (and the corresponding host component), see “type attribute” on page 102.

2. HATS instantiates a widget object based on the **widget** attribute setting.

   If HATS recognizes the value of the **widget** attribute as one of its widgets, HATS instantiates the appropriate widget. If HATS does not recognize the value of the **widget** as one of its widgets, HATS attempts to instantiate the custom widget. You must specify the fully qualified path, like `com.company.division.product.MyWidget`, and place the class file in either the `WEB-INF/classes` directory or in a jar file in the `WEB-INF/lib` directory. For example, if the **widget** attribute is set to Link, then the `com.ibm.hats.widget.LinkWidget` class is instantiated.

   For a list of valid values for the **widget** attribute (and the corresponding widget), see “widget attribute” on page 102.

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

   Each component has a different implementation of the `recognize()` method. HATS uses the method to initialize several parameters (such as HostScreen, start row, start col, end row, and end col). The `recognize()` method also performs pattern recognition logic and host screen data location. By default, a class named `ComponentElementPool`, which contains several ComponentElement objects, stores host screen data that the `recognize()` method locates.

4. HATS performs text replacement on the `ComponentElementPool` returned by the component's `recognize()` method.
5. HATS invokes the widget's `draw()` method.

   The `draw()` method renders the component as HTML output. The widget passes the writer object to write HTML to the browser. The `draw()` method passes the `ComponentElementPool` (from Step 3 on page 56) to the widget. The HTML output is based on the host screen data stored in a `ComponentElementPool` object.

## Creating a custom host component

You can customize host components by creating a new host component or modifying an existing host component. If you want to create a new host component, you must extend the host component abstract class, `com.ibm.hats.component.ComponentExtract`, and overwrite the `recognize()` method. If you want to modify an existing host component, you must extend an existing host component class and overwrite its `recognize()` method.

HATS provides the following host component classes of abstract class `com.ibm.hats.component.ComponentExtract`.

- com.ibm.hats.component.CommandLineExtract
- com.ibm.hats.component.FieldExtract
- com.ibm.hats.component.FieldTableExtract
- com.ibm.hats.component.FunctionKeyExtract
- com.ibm.hats.component.DefaultExtract
- com.ibm.hats.component.InputFieldExtract
- com.ibm.hats.component.MenuExtract
- com.ibm.hats.component.SelectionListExtract
- com.ibm.hats.component.SubfileExtract
- com.ibm.hats.component.TextExtract
- com.ibm.hats.component.VisualTableExtract

When HATS runs a project, it instantiates the custom host component based on the setting of the **type** attribute of the HATS:Component tag.

The host component class inherits the following methods from the parent class without implementation:

**public ComponentElementPool recognize(HostScreen hostScreen, int startRow, int startCol, int endRow, int endCol, String label, Properties settings)**

The `recognize()` method initializes many of the data members that are needed by this class to perform pattern recognition and gathers host screen data from the HostScreen object. This method has a different implementation in each host component class. You should overwrite this method to implement your own pattern recognition logic.

In addition to initializing variables, this method also instantiates an object named `hostComponentData` (data type `com.ibm.hats.common.ComponentElementPool`) used to store the host screen data gathered. `ComponentElementPool` is a container class carrying a vector of ComponentElement objects and other needed host screen information (such as cursor position). ComponentElement describes the general information of host components that widget classes can use to render in HTML.

For a description of the arguments of this method, see the Java documentation for the `recognize()` method of the `ComponentExtract` class at the product Web site (http://www.ibm.com/software/webservers/hats).

## Creating a custom widget

You can customize widgets by creating a new widget or modifying an existing widget. If you want to create a new widget, you must extend the abstract widget parent class, `com.ibm.hats.widget.Widget`, and overwrite the abstract `draw()` method. If you want to modify an existing widget, you must extend one of the following existing widget classes and overwrite one of its methods.

- com.ibm.hats.widget.ButtonWidget
- com.ibm.hats.widget.ButtonTableWidget
- com.ibm.hats.widget.DefaultWidget
- com.ibm.hats.widget.DropDownListWidget
- com.ibm.hats.widget.FieldWidget
- com.ibm.hats.widget.HorizontalBarGraphWidget
- com.ibm.hats.widget.LabelWidget
- com.ibm.hats.widget.LineGraphWidget
- com.ibm.hats.widget.LinkWidget
- com.ibm.hats.widget.OptionListWidget
- com.ibm.hats.widget.SubfileWidget
- com.ibm.hats.widget.TableWidget
- com.ibm.hats.widget.TextInputWidget
- com.ibm.hats.widget.VerticalBarGraphWidget

When HATS runs a project, it instantiates the custom widget based on the setting of the **widget** attribute of the HATS:Component tag.

The widget class inherits the following methods from the parent class without implementation:

**public void draw(java.io.Writer out, Object o, Properties widgetSettings)**
The `draw()` method first initializes data members that will be needed to write the HTML code out. Each widget class has its own implementation of the `draw()` method. You should overwrite this method to create a custom widget.

For a description of the arguments of this method, see the Java documentation for the `draw()` method of the `Widget` class at the product Web site (http://www.ibm.com/software/webservers/hats).

## Registering your component or widget

After creating a custom component or widget, you must import the source code for the component or widget into the Source folder of your project. Select **File > Import > File System** to open the Import wizard. In the Import wizard, select the location of your source files in the **Directory** field. Select the Source folder (or a package in the Source folder) of your project in the destination **Folder** entry field, and ensure that the **Create complete folder structure** checkbox is not checked. When your source .java files are imported, they are automatically compiled as .class files and packaged into your HATS project in the `webApplication/WEB-INF/classes/` directory path of the **Navigator** tab of the HATS Studio.

Host components must map to specific widgets. Custom host components can map to any existing widget or to a custom widget. After you import your source code for a custom component or widget into the project's Source folder, you need to edit the `ComponentWidget.xml` file to add your custom component and associate defined widgets or to associate your custom widgets with components. If you are only adding a custom widget, you must associate the custom widget with a defined component.

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

To edit the `ComponentWidget.xml` file, click the **Navigator** tab of the HATS Studio. The `ComponentWidget.xml` file is shown at the bottom of the **Navigator** view of your project. The following is a sample of the `ComponentWidget.xml` file that shows the HATS-supplied visual table component and one of the associated widgets, the vertical bar graph widget.

```
<ComponentWidgetList>
    <components>
        <component className="com.ibm.hats.component.VisualTableExtract"
                    displayName="%VISUAL_TABLE_COMPONENT">
            <associatedWidgets>
                <widget className="com.ibm.hats.widget.VerticalBarGraphWidget"/>
            </associatedWidgets>
        </component>
    </components>


    <widgets>
        <widget className="com.ibm.hats.widget.VerticalBarGraphWidget"
                  displayName="%VERTICAL_BAR_GRAPH_WIDGET"/>
    </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 Studio, 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 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 Studio. This name must be unique among the registered widgets. The form of the displayName for a custom widget is simply a string, without the percent sign (%). Spaces are not allowed in the displayName. However, you can use an underscore ( _ ) in place of a space.

**widget**
Identifies the widgets 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 only contains 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, and make it available to the HATS Studio, 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 recognize the component. 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 Studio. This name must be unique among the registered widgets. The form of the displayName for a custom widget is simply a string, without the percent sign (%). Spaces are not allowed in the displayName. However, you can use an underscore ( _ ) in place of a space.

## HATS Studio support for custom components and widgets

HATS Studio can support custom components and widgets as well as existing components and widgets. The graphical user interface (GUI) for inserting or editing the components and widgets is built dynamically based on information provided by the component and widget classes. If you want the GUI to support your custom component or widget, you can override the `getPropertyPageCount()` method, and must override the `registerProperties()` method.

**public int getPropertyPageCount()**
The `getPropertyPageCount()` method determines how many wizard pages are needed in HATS Studio to customize all of the object's attributes. If you do not override this method, only one wizard page is used to customize all of the attributes. Overriding this method allows you more control over the GUI. For example, you might want to show customizable attributes on more than one page, or some of the attributes might depend on the values of other attributes.

The return value for this method is as follows:

**int** The number of wizard pages needed in HATS Studio to fully customize all the attributes of this object.

**public Vector getCustomProperties(int iPageNumber, Properties properties, ResourceBundle bundle)**
The `getCustomProperties()` method returns a vector of **HCustomProperty** customizable property objects. A customizable property is any property of the object that can be edited by a HATS Studio project developer using this component or widget object in a transformation. The components and widgets use the customizable properties to determine what host information to look for, or to determine how data should be displayed.

For a description of the arguments of this method, see the Java documentation at the product Web site (http://www.ibm.com/software/webservers/hats) for the `getCustomProperties()` method of the `Component` or `Widget` class.

**public HCustomProperty(String name, int type, String label, boolean isRequired, String[] prefilledValues, String[] prefilledCodes, String defaultValue,**

**ICustomPropertyValidator validator, String helpID)**

This is the constructor for a customizable property object.

For a description of the arguments of this constructor, see the Java documentation at the product Web site (http://www.ibm.com/software/webservers/hats) for the `HCustomProperty` class.

# Chapter 15. Administering HATS applications

The HATS Administration Console enables the system administrator to view sessions initiated for HATS applications. The panels in HATS administration display license usage information and session information for applications included in an .ear file. Each application in an .ear file includes files to support HATS administration.

To start HATS Administration Console, enter the following URL in your Web browser:

`http://localhost/appname/HATSAdmin/admin.jsp`

where *localhost* is the hostname or IP address of the machine where your HATS applications are deployed, and *appname* is the name of an application in the .ear file.

More than one HATS Administration Console can be started using different application names included in an .ear file. The panel information in each console displays the same license usage and session information for the set of applications contained in the .ear file. The session information can be sorted by application name, connection identifier, or communication status by clicking the headings above the session information. On this panel, you can shut down one or more connections being used by the HATS applications included in the .ear file. You should encourage your end users to click **Disconnect** to terminate their session rather than simply closing the browser window, because sessions stay active for 30 minutes when not terminated properly. This invalidation timeout value is set in WebSphere Application Server (WAS).

HATS Administration Console is bound to WebSphere security. If WebSphere security is active on WAS, and the URL to start HATS Administration Console is entered in a browser, the page is redirected to a HATS Administration Console login authentication page. You must enter the user ID and password for the WAS security server before HATS Administration Console can be started.

If WebSphere is running in a cloned environment, HATS Administration Console does not display all the connections for all clones. To display the connections for a specific clone, enter the following URL in your Web browser:

`http://localhost/appname/HATSAdmin/selectclone.jsp`

where *localhost* is the hostname or IP address of the machine where your HATS applications are deployed, and *appname* is the name of an application in the .ear file. This Web page enables you to select the clone for which you want to view information. To specify the individual clone, you need to know the session ID value of the connection for an end user. You need to request that information from the end user. The end user can find the session ID by viewing the source of their application Web page, and searching for the following string:

`<INPUT TYPE="HIDDEN" NAME="SESSIONID" VALUE="value" />`

where *value* appears like *0000***YGDDXENHPWS3XANVO2LUR3Y:**tu3bu7f3. Ignore the *0000* in the *value*. The **YGDDXENHPWS3XANVO2LUR3Y** is the connection identifier, and the tu3bu7f3 is the clone ID.

**Note:** The entire *value* is not displayed in HATS Administration Console panel displays; only the **YGDDXENHPWS3XANVO2LUR3Y** appears in the display.

When you receive the *value* from your end user, insert it into the **Session ID** field of the of `selectclone.jsp` page and click **Submit** to display the HATS Administration Console for the specific clone.

# Chapter 16. Troubleshooting HATS

This chapter provides information that enables you to use message logs and traces to diagnose problems that could occur when users interact with your HATS application. It also outlines some problems that could be encountered, and solutions to those problems.

## Message logs and traces

When a HATS application runs in the WebSphere Application Server, logging and tracing is performed for the application. Logging and tracing is controlled by the settings of the `runtime.properties` file. The location of this file differs between the machine where you installed HATS Studio and the server machine where your applications run:

**HATS Studio (for the Run on Server function)**
HATS Studio files are stored on your system under the drive and directory where you installed your WebSphere Studio program, such as WebSphere Studio Application Developer. In the `workspace` subdirectory, a folder with the Enterprise Application project name you specified when you created your project resides. The path to the `runtime.properties` file is `/workspace/fn.ear/runtime.properties`, where *fn* is the Enterprise Application project name. You can also locate the file on the **Navigator** tab of the HATS Studio under the `/fn.ear/` folder. Double-click the `runtime.properties` file to open the file in a HATS Studio editor.

**Server** In the drive and directory where you installed WebSphere, there is a folder named `installedApps`. In the `installedApps` subdirectory, a folder with the Enterprise Application project name you specified when you created your project resides. The path to the `runtime.properties` file is `/installedApps/fn.ear/runtime.properties`, where *fn* is the Enterprise Application project name.

You can use any text editor to modify the `runtime.properties` file. If you modify the file, you must stop and restart the server that is running the application for the changes to take effect.

The `runtime.properties` file contains the following basic properties:

**Note:** Names of the properties are case-sensitive. Do not change the property names.

**num_licenses**
Specifies the number of licenses you purchased. HATS tracks the number of HATS connections to host resources and logs a message when the value exceeds the number of licenses purchased.

The value is an integer. There is no default. Specify num_licenses = – 1 if you purchased an unlimited license.

**licenseTracking**
Specifies whether HATS tracks license usage or not. The value is binary. The default is 0.

**0** HATS does not track license usage.

**1** HATS tracks license usage for all application servers in a node. The license usage information is written to a file named licensex.txt in the log directory of the HATS application installation directory on the server, where the x is either 1 or 2.

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

The license usage files contain the following information, arranged in rows, with each row representing one hour of operation. The values are separated by a space ( ).

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

**maxTraceFiles**

The maximum number of trace information files. The default is 5.

The base trace file name in runtime.properties is used as a template to generate unique sets of trace files for each application server. The default base name for a trace file is *trace.txt*, which can be changed.

An index (1, 2, 3, and so forth) is added to this name to distinguish multiple trace files. When *trace1.txt* reaches maxTraceFileSize, it is closed and renamed to *trace2.txt*. A new *trace1.txt* file is opened.

When the maxTraceFiles number is exceeded, the oldest file is deleted.

The location of the *trace*.txt* files differs between the machine where you installed HATS Studio and the server machine where your applications run:

**HATS Studio (for the Run on Server function)**
HATS Studio files are stored on your system under the drive and directory where you installed your WebSphere Studio program, such as WebSphere Studio Application Developer. In the `workspace` subdirectory, a folder with the Enterprise Application project name you specified when you created your project resides. In the Enterprise Application project name directory, there is a `/logs` directory. The path to the *trace*.txt* files is `/workspace/`*`fn`*`.ear/logs/trace*.txt`, where *fn* is the Enterprise Application project name.

**Server** In the drive and directory where you installed WebSphere, there is a folder named `installedApps`. In the `installedApps`subdirectory, a folder with the Enterprise Application project name you specified when you created your project resides. The path to the *trace*.txt* files is `installedApps/`*`fn`*`.ear/logs/trace*.txt`, where *fn* is the Enterprise Application project name.

**maxTraceFileSize**

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

The value is a decimal integer. The default is 1024 KB.

**traceFile**
The name used as a template to generate file names for each set of application server files to which trace messages are written. The default base name for a trace file is *trace.txt*.

**maxLogFiles**
The maximum number of message files. The default is 2.

The base message log file name in runtime.properties is used as a template to generate unique sets of message log files for each application server. The default base name for a log file is *messages.txt*, which can be changed.

An index (1, 2, 3, and so forth) is added to this name to distinguish multiple message log files. When *messages1.txt* reaches maxLogFileSize, it is closed and renamed to *messages2.txt*. A new *messages1.txt* file is opened.

When the maxLogFiles number is exceeded, the oldest file is deleted.

The location of the *messages*.txt* files differs between the machine where you installed HATS Studio and the server machine where your applications run:

**HATS Studio (for the Run on Server function)**
HATS Studio files are stored on your system under the drive and directory where you installed your WebSphere Studio program, such as WebSphere Studio Application Developer. In the `workspace` subdirectory, a folder with the Enterprise Application project name you specified when you created your project resides. In the Enterprise Application project name directory, there is a `/logs` directory. The path to the *messages*.txt* files is `/workspace/`*`fn`*`.ear/logs/messages*.txt`, where *fn* is the Enterprise Application project name.

**Server** In the drive and directory where you installed WebSphere, there is a folder named `installedApps`. In the `installedApps`subdirectory, a folder with the Enterprise Application project name you specified when you created your project resides. The path to the *messages*.txt* files is`installedApps/`*`fn`*`.ear/logs/messages*.txt`, where *fn* is the Enterprise Application project name.

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

The value is a decimal integer. The default is 512 KB.

**logFile**
The name used as a template to generate file names for each set of application server files to which log messages are written. The default base name for a trace file is *messages.txt*.

The runtime.properties file contains the following HATS server tracing properties:

**trace.RUNTIME**
Specifies the level of tracing for the main runtime and for all settings under RUNTIME.* that do not specify a trace level.

The value is an integer from 0–9. The default is 0.

See the description of the tracelevel.* keys for information on values for this setting.

**trace.RUNTIME.WIDGET**
Specifies the level of tracing for HATS widgets. This setting overrides the setting of trace.RUNTIME for tracing of widgets.

The value is an integer from 0–9. The default is 0.

See the description of the tracelevel.* keys for information on values for this setting.

**trace.RUNTIME.ACTION**
Specifies the level of tracing for HATS event actions. This setting overrides the setting of trace.RUNTIME for tracing of event actions.

The value is an integer from 0–9. The default is 0.

See the description of the tracelevel.* keys for information on values for this setting.

**trace.RUNTIME.COMPONENT**
Specifies the level of tracing for HATS components. This setting overrides the setting of trace.RUNTIME for tracing of components.

The value is an integer from 0–9. The default is 0.

See the description of the tracelevel.* keys for information on values for this setting.

**trace.UTIL**
Specifies the level of tracing for HATS runtime utilities.

The value is an integer from 0–9. The default is 0.

See the description of the tracelevel.* keys for information on values for this setting.

**tracelevel.x**
Each of the tracelevel keys specifies as its value a hexadecimal digit string. This string is a mask which is applied to the tracing feature for components which use that trace level. Each bit of the digit string controls one type of tracing for HATS.

The values of tracelevel.1 through tracelevel.7 should not be changed unless requested by IBM support. Otherwise, specifying these seven tracelevel.* properties is not necessary.

Tracelevel.8 and tracelevel.9 values can be used to create customized tracing levels.

The default values are:

**tracelevel.1**
0000000000020000

**tracelevel.2**
000000000000020f

**tracelevel.3**
000000000004023f (minimum)

**tracelevel.4**
0000000000041a3f

**tracelevel.5**
00000000000c1bbf (normal)

**tracelevel.6**
00000000000c1bbf

**tracelevel.7**
00000000001c1bbf (maximum)

**tracelevel.8**
00000000001c1bbf

**tracelevel.9**
00000000001c1bbf

To customize the trace masks, add together the following (hex) values:

**x000001**
Informational messages

**x000002**
Warning messages

**x000004**
Error messages

**x000008**
Critical error messages

**x000010**
API traces

**x000020**
Callback API traces

**x000080**
Method entry

**x000100**
Method exit

**x000200**
Exceptions

**x000400**
Miscellaneous traces

**x000800**
Object creation

**x001000**
Object disposal

**x020000**
Reserved

**x040000**
Miscellaneous data - level 1

**x080000**
Miscellaneous data - level 2

**x100000**
Miscellaneous data - level 3

The runtime.properties file contains the following Host On-Demand tracing properties:

**Note:** You should not enable the Host On-Demand traces (except for PSEVENT, OIAEVENT, and COMMEVENT) unless requested by IBM Support.

**trace.HOD.PS**
Specifies the level of Host On-Demand presentation space tracing.

The value is an integer from 0–3. The default is 0.

**trace.HOD.DS**
Specifies the level of Host On-Demand data stream tracing.

The value is an integer from 0–3. The default is 0.

**trace.HOD.TRANSPORT**
Specifies the level of Host On-Demand transport tracing.

The value is an integer from 0–3. The default is 0.

**trace.HOD.MACRO**
Specifies the level of tracing for Host On-Demand macros.

The value is an integer from 0–2. The default is 0.

**0** Host On-Demand macro tracing is not enabled.

**1** Host On-Demand event tracing is enabled.

**2** Host On-Demand support tracing is enabled.

**trace.HOD.USERMACRO**
Specifies the level of tracing for trace actions in Host On-Demand macros.

The value is an integer from 0–3. The default is 0.

**trace.HOD.SESSION**
Specifies the level of Host On-Demand session tracing.

The value is an integer from 0–3. The default is 0.

**trace.HOD.PSEVENT**
Specifies the level of Host On-Demand PS events.

The value is an integer from 0–1. The default is 0.

**trace.HOD.OIAEVENT**
Specifies the level of Host On-Demand OIA events.

The value is an integer from 0–1. The default is 0.

**trace.HOD.COMMEVENT**
Specifies the level of Host On-Demand COMM events.

The value is an integer from 0–1. The default is 0.

# Problems and solutions

This section describes problems that could occur when users interact with your HATS application, and solutions to those problems.

## Incorrect data in HATS applications with non-English locales

HATS application input can be corrupted in non-English locales and result in incorrect data being used.

HATS applications rely on input data from Servlet and Java Server Page (JSP) API's to retrieve HTML FORM data from the FORM character set and convert it to Unicode. The Servlet's getParameter() methods must decide on the character set of

the FORM. HATS applications create JSPs in all locales using the UTF-8 character set; so UTF-8 is the required character set for processing HATS application data.

The specification for FORM provides a `charset` value in the `content-type` attribute, but most browsers do not add the `charset` value to `content-type`. Also, in WebSphere Application Server 4.0 (WAS), there is no way for the HATS application to dynamically specify the character set used for each form; however, you can assign a character set mapping used by the entire WAS. A customizable properties file provides a locale-to-character set mapping used by WAS. The file is `\\WebSphere\AppServer\properties\encoding.properties`.

The `encoding.properties` file determines which character set is used in reading input data. For example:

```
en=ISO-8859-1
fr=ISO-8859-1
.
.
.
zh=GB2312
zh_TW=Big5
```

The default ISO-8859 character set works in most cases; however, input data in non-Latin1, double-byte, and bi-directional locales is frequently corrupted.

To establish UTF-8 as the character set, edit the `encoding.properties` file.

1. In this file, find the entry for the WAS locale. For example, Traditional Chinese is `zh_TW` and Simplified Chinese is `zh`.
2. Change the value of this entry to UTF-8 for that locale. For example, `zh=UTF-8`.
3. Save your changes, then restart WAS.
4. You should now be able to run your internationalized HATS application.

## Thai font size too small for default transformation

The default font size for HATS applications is 10–point type. For a Thai session, the default font size is too small; it should be 12–point type. For transformations to display well on a Thai session, you should increase the font size by applying one of the supplied stylesheets or modyfing the stylesheet you are using to display the proper font size.

## End users receiving HTTP 404 error

If the end users of your application are receiving an `HTTP 404 - File not found when trying to run an application` message, WebSphere Application Server could be configured incorrectly.

If your deployed .ear file contains the .war files for multiple applications, the WebSphere Application Server (WAS) administrator must configure the server with **Module Visibility** set to **Application**.

# Chapter 17. Messages reference

The messages in this chapter are those that can be issued when your HATS application runs on the WebSphere Application Server. Messages in HATS Studio do not have message numbers; error conditions in HATS Studio are displayed at the top of the wizard panels.

**HAT0001** **Loading HATS configuration data from {0}.**

**Explanation:** {0} is the filename of the configuration data file.

**Response:** None.

**HAT0002** **The HATS message log file is {0}.**

**Explanation:** {0} is the filename of the message log file.

**Response:** None.

**HAT0003** **The HATS trace log file is {0}.**

**Explanation:** {0} is the filename of the trace log file.

**Response:** None.

**HAT0040** **A program exception occurred. There may be additional messages in this log which describe the error. The following information may help determine the cause, if this log is requested by IBM service:**

**{0}**
**{1}**

**Explanation:** {0} is the exception message string as received by Java.

{1} is the exception stack trace.

**Response:** Check the log for additional messages.

**HAT0060** **File {0} was not found.**

**Explanation:** {0} is the filename of the missing file.

**Response:** Contact IBM service for assistance.

**HAT0061** **Directory {0} was not found.**

**Explanation:** {0} is the missing directory.

**Response:** Contact IBM service for assistance.

**HAT0062** **Archive file {0} could not be created.**

**Explanation:** {0} is the filename of the archive file.

**Response:** Ensure that the application's connection settings, specifically the SSL certificate setting, are correct. Contact IBM service for assistance.

**HAT0300** **An unexpected exception was received: {0}**

**Explanation:** {0} is the exception.

**Response:** If the exception is issued by code written by a user, such as business logic or custom components or widgets, contact the programmer who wrote the code. Otherwise, contact IBM service for assistance.

**HAT0350** **An error occurred reading the file {0}.**

**Explanation:** {0} is the filename of the file being read.

**Response:** Ensure that the file exists and is a valid file. Contact IBM service for assistance.

**HAT0351** **An error occurred writing the file {0}.**

**Explanation:** {0} is the filename of the file being written.

**Response:** Ensure that the filename is valid. Contact IBM service for assistance.

**HAT0352** **Cannot find the file named {0}.**

**Explanation:** {0} is the filename of the missing file.

**Response:** Restore the missing file.

You might need to rebuild the HATS project before the application is run.

**Note:** You can set your WebSphere Studio workbench preferences to perform a build automatically when a resource has been modified.

**HAT0353** **Creating the file {0}.**

**Explanation:** {0} is the filename of the file being created.

**Response:** This is an informational message. No response is required.

**HAT0354 Failed to delete the file named {0}.**

**Explanation:** {0} is the filename of the file that could not be deleted.

**Response:** Contact IBM service for assistance.

**HAT0360 Licenses used ({0}) exceeding licenses purchased ({1}).**

**Explanation:** {0} is the number of licenses used.

{1} is the number of licenses purchased.

**Response:** Verify that the number of licenses you configured is correct.

**HAT0400 An exception occurred creating the host configuration. A message containing the exception details follows.**

**Response:** Use the information in the message that follows to bypass the exception during host configuration.

**HAT0401 Cannot connect to the host using the following session properties:**

**{0}**

**Explanation:** {0} is a set of session properties.

**Response:** Examine the properties to verify that the named server exists and is available.

**HAT0402 Session is in CONNECTION_ACTIVE state, but not CONNECTION_READY state. A possible reason could be that the TN server port specified does not support the data stream expected.**

**Session properties being used: {0}**

**Explanation:** {0} is a set of session properties.

**Response:** Ensure that the connection type (TN3270, TN3270E, TN5250) is correct.

**HAT0403 Cannot load the application {0}.**

**Explanation:** The `application.hap` file could not be loaded for the named application.

{0} is the name of the application.

**Response:** The required file could not be located or has been corrupted. Republish the application and redeploy the .ear file. Tracelevel.3 runtime traces will display the expected filename and location.

**HAT0405 Session is ready, but no host data was available. Session properties:**

**{0}**

**Explanation:** {0} displays the session properties.

The host connection came up, but the host sent no screen data.

**Response:** Ensure that the application's connection settings are correct. Contact IBM service for assistance.

**HAT0410 An error occurred while processing this request.**

**Detailed error information: {0}**

**Explanation:** {0} is the exception stack trace or other application programmer information.

This message might be followed by another numbered message with further information.

**Response:** If the application is still available, you might be able to continue the application. To determine what caused the error, check the other messages appearing with this message and check the message log or trace file for additional information.

**HAT0411 The host connection is inactive.**

**Explanation:** The host became inactive during the processing of the application.

**Response:** If the application is ending normally, no response is necessary. If the application ends abnormally, check the message log for additional information. Also, cnsure that the application's connection settings are correct. Additional information might be found at the Telnet server.

**HAT0412 An error occurred while processing an application Web page ″{0}″.**

**Explanation:** {0} is the name of the Web page (such as a transformation .jsp) being processed.

**Response:** Contact IBM service for assistance.

**HAT0600 An error occurred while retrieving the action list for the screen customization ″{0}″.**

**Explanation:** {0} is the name of the screen customization.

**Response:** Rebuild the HATS project before the application is run.

**Note:** You can set your WebSphere Studio workbench preferences to perform a build automatically when a resource has been modified.

**HAT0601** **The action {0} specifies an invalid parameter.**

**Explanation:** {0} is the name of the action.

**Response:** Specify a valid parameter for the action.

**HAT0602** **Cannot insert null value to the host screen at row {0}, column {1}.**

**Explanation:** {0} is a row number on the host screen.

{1} is a column number on the host screen.

**Response:** Ensure that the variable inserted onto the host screen has a value.

**HAT0603** **Cannot insert the value {0} to the host screen at row {1}, column {2}, because the host screen size is {3}.**

**Explanation:** {0} is the value.

{1} is a row number on the host screen.

{2} is a column number on the host screen.

{3} is the host screen size.

**Response:** Insert a value at the row and column location that is valid for the host screen size.

**HAT0604** **An exception occurred while calculating the value for the global variable {0}. The action which failed was {1}. A message containing the exception details follows.**

**Explanation:** {0} is the name of the global variable.

{1} is the name of the action that failed.

**Response:** Use the information in the message that follows to bypass the exception during calculation of the value for the global variable.

**HAT0605** **An error occurred while processing the actions for the screen customization ″{0}″.**

**Explanation:** {0} is the name of the screen customization.

**Response:** Contact IBM service for assistance.

**HAT0606** **Global variable ″{0}″ does not exist.**

**Explanation:** {0} is the name of the global variable.

**Response:** Check the message log for additional information. Examine the trace file, if one exists. Check the application's screen customization actions to ensure that all required global variables are created.

**HAT0607** **The index {0} for global variable ″{1}″ is out of bounds. The global variable has {2} elements.**

**Explanation:** {0} is the value of the global variable index.

{1} is the name of the global variable.

{2} is the current size of the global variable.

**Response:** Check the message log for additional information. Examine the trace file, if one exists. Check the application's screen customization actions to ensure that all required global variables are created and contain enough elements to complete the failing action.

**HAT0608** **Cannot insert the value ″{0}″ to the host screen at row {1}, column {2}, because the location is not contained in an unprotected field.**

**Explanation:** {0} is the name of the global variable.

{1} is a row number on the host screen.

{2} is a column number on the host screen.

**Response:** Choose a valid row and column location for the currentl screen. Ensure that the global variable is being inserted into an unprotected field.

**HAT0700** **The attribute ″{0}″ of widget setting does not exist.**

**Explanation:** {0} is the attribute in the widget setting that does not exist.

**Response:** Specify a valid attribute in the widget setting.

**HAT0701** **The value of attribute ″{0}″ in widget setting is empty.**

**Explanation:** {0} is the attribute of the widget setting that is empty.

**Response:** Specify a value for the attribute in the widget setting.

**HAT0702** **The value of attribute ″{0}″ in widget setting is invalid.**

**Explanation:** {0} is the attribute of the widget setting that is invalid.

**Response:** Specify a valid value for the attribute in the widget setting.

**HAT0800** **An exception occurred setting the property of print session. A message containing the exception details follows.**

**Response:** Use the information in the message that follows to bypass the exception that occurred while

setting the property of the print session.

---

**HAT0801** **An exception occurred processing PDF I/O to client browser. A message containing the exception details follows.**

**Response:** Use the information in the message that follows to bypass the exception during processing of the PDF I/O to the client browser.

# Chapter 18. Language support

The HATS user interface and context-sensitive help are provided in these languages:

- Brazilian Portuguese
- English
- French
- German
- Italian
- Japanese
- Korean
- Simplified Chinese
- Traditional Chinese

All the languages are installed in a single product image. National language support is operating-system dependent, so the appropriate font and keyboard support for the language you want to use must be installed in the operating system. For example, if you want to use French as the host-session language but do not have the French font and keyboard support installed, you may not be able to display the correct characters.

HATS supports the following code pages. You can choose the code page for each HATS project when you create the project, and you can modify it later in the project editor.

*Table 2. Code pages*

| Code page | Location or usage |
|---|---|
| 037 | Belgium<br>Brazil<br>Canada<br>Netherlands<br>Portugal<br>United States |
| 273 | Austria<br>Germany |
| 274 | Belgium (Old) |
| 275 | Brazil (Old) |
| 277 | Denmark<br>Norway |
| 278 | Finland<br>Sweden |
| 280 | Italy |
| 284 | Spain<br>Latin-America (Spanish) |
| 285 | United Kingdom |
| 290 | Japan (Katakana Extended) |
| 297 | France |

*Table 2. Code pages (continued)*

| Code page | Location or usage |
|---|---|
| 420 | Arabic Speaking |
| 424 | Hebrew (New Code) |
| 500 | Multilingual |
| 803 | Hebrew (Old Code) |
| 838 | Thai |
| 870 | Bosnia/Herzegovina<br>Croatia<br>Czech Republic<br>Hungary<br>Poland<br>Romania<br>Slovakia<br>Slovenia |
| 871 | Iceland |
| 875 | Greece |
| 924 | Multilingual ISO Euro |
| 930 | Japanese (Katakana) |
| 933 | Korea |
| 937 | Taiwan (Traditional Chinese) |
| 939 | Japanese (English) |
| 1025 | Belarus<br>Bulgaria<br>FYR Macedonia<br>Russia<br>Serbia/Montenegro (Cyrillic) |
| 1026 | Turkey |
| 1047 | Open Edition |
| 1112 | Latvia<br>Lithuania |
| 1122 | Estonia |
| 1123 | Ukraine |
| 1137 | Hindi |
| 1140 | Belgium Euro<br>Brazil Euro<br>Canada Euro<br>Netherlands Euro<br>Portugal Euro<br>United States Euro |
| 1141 | Austria Euro<br>Germany Euro |
| 1142 | Denmark Euro<br>Norway Euro |
| 1143 | Finland Euro<br>Sweden Euro |
| 1144 | Italy Euro |

*Table 2. Code pages (continued)*

| Code page | Location or usage |
|---|---|
| 1145 | Latin-America Euro (Spanish)<br>Spain Euro |
| 1146 | United Kingdom Euro |
| 1147 | France Euro |
| 1148 | Multilingual Euro |
| 1149 | Iceland Euro |
| 1153 | Bosnia/Herzegovina Euro<br>Croatia Euro<br>Czech Republic Euro<br>Hungary Euro<br>Poland Euro<br>Romania Euro<br>Slovakia Euro<br>Slovenia Euro |
| 1154 | Belarus Euro<br>Bulgaria Euro<br>FYR Macedonia Euro<br>Russia Euro<br>Serbia/Montenegro (Cyrillic) Euro |
| 1155 | Turkey Euro |
| 1156 | Latvia Euro<br>Lithuania Euro |
| 1157 | Estonia Euro |
| 1158 | Ukraine Euro |
| 1160 | Thai Euro |
| 1364 | Korea Euro |
| 1371 | Taiwan (Traditional Chinese) Euro |
| 1388 | PRC (Simplified Chinese Extended; GB18030) |
| 1390 | Japanese (Katakana Unicode Extended) |
| 1399 | Japanese (Latin Unicode Extended) |

# Chapter 19. Bi-directional application support

This chapter explains how to use the functions provided by HATS for developing applications in bi-directional languages. Generally its contents apply to both Hebrew and Arabic application developers. The functions that are specific to Arabic users are described separately.

HATS provides these functions to support bi-directional languages:

- Support of bi-directional code pages
- Correct bi-directional text processing and symmetric swapping
- Support of left-to-right and right-to-left screen customization
- A **Screen Reverse** button to toggle between left-to-right (LTR) and right-to-left (RTL) screen orientation
- Different levels of user control over screen orientation
- Control over the orientation of host components, widgets, and text, which may be opposite to general screen orientation
- Visual Input Field support
- Bi-directional text in global variables and text replacement, including different algorithms for text replacement in screens with left-to-right and right-to-left orientation.

This chapter explains all these features.

## Software environment

The following are required for bidirectional application support:

- The supported browser and its version is Internet Explorer version 5.0 or higher.
- The default locale of the machine where WebSphere Studio is installed should be set to Arabic for Arabic users and Hebrew for Hebrew users.
- The default locale for the end user client machine must be set to Arabic for Arabic users and Hebrew for Hebrew users.
- For data input in bi-directional code pages to be processed correctly, UTF-8 must be specified for Hebrew and Arabic locales in `WebSphere\AppServer\encoding.properties` on the machine running the WebSphere Application Server. See "Incorrect data in HATS applications with non-English locales" on page 70 for more information.

## Working with the host terminal

The HATS host terminal allows Host On-Demand bi-directional-specific keystrokes, so you can perform the following bi-directional functions. The following host function keys are available for both 3270 and 5250 sessions:

**Ctrl+L: Latin Layer**
This key combination changes the language layer to Latin and the operator information area (OIA) is updated to show English Language.

**Ctrl+S: Screen Reverse**
If the screen orientation is left-to-right, this key combination changes the screen image to right-to-left and the language layer changes to Bidi. If the

screen orientation is right-to-left, this key combination reverses the screen image to left-to-right and the language changes to Latin.

**Ctrl+N: Bidi Layer**
This key combination changes the language layer to Bidi and the OIA is updated to show Bidi Language.

**Ctrl+F: Field Reverse**
If the field orientation is left-to-right, this key combination changes the field orientation to right-to-left, the cursor moves to the other side of the field, and the language layer becomes Bidi. If the field orientation is right-to-left, this key combination changes the field orientation to left-to-right, the cursor moves to the other side of the field, and the language layer becomes Latin.

The following host function keys are available only for 3270 sessions:

**Ctrl+P: Push**
You can enter and edit text in the opposite direction from the field direction.

**Ctrl+O: End Push**
Push mode is ended and the cursor moves to the end of the push segment.

**Ctrl+A: Auto Push**
You can type mixed left-to-right and right-to-left text by changing the language layer.

The following host function key is available only for 5250 sessions:

**Ctrl+C: Close**
The data entered in one keystroke direction (either left-to-right or right-to-left) is concatenated with the data that was previously entered in the opposite direction. The cursor direction is set to be the same as the field direction, and the language layer is set to the default for the field direction. If the screen orientation is currently left-to-right, the cursor is positioned at the first null to the right of the concatenated text. If the screen orientation is currently right-to-left, the cursor is positioned at the first null to the left of the concatenated text.

### Capturing screens

In bi-directional sessions, screens can be captured either as left-to-right screens or as right-to-left screens. Captured screens are displayed exactly as they were captured. To capture a screen as a right-to-left screen, press **Ctrl+S** (Screen Reverse) in a left-to-right screen and click **Create Screen Capture**.

## Recognizing bi-directional host components

Usually the orientation of a host component is the same as the screen orientation. However, in some cases, the orientation of a host component is the opposite of the general orientation of the screen. When a HATS project uses a bi-directional code page, a check box labeled **Component orientation opposite to screen orientation** is added to the Insert Host Component wizard. When you add a host component to a transformation, check this box if the host component's orientation is the opposite of the screen orientation. Checking the box enables HATS to recognize command prompts, function keys, selection lists, and menus whose orientation is the opposite of the screen orientation.

## Controlling the orientation of widgets

There are two special check boxes that provide you, the BIDI application developer, control over widget presentation when your application runs in the WebSphere Application Server:

- Widget Orientation opposite to Screen Orientation
- Text Orientation opposite to Screen Orientation.

On a left-to-right screen, when the Widget Orientation check box is selected, the GUI image is changed to be right-to-left. Similarly, when the Text Orientation check box is selected, the text within the GUI image is changed to left-to-right. By default, selection of the Widget Orientation check box causes the Text Orientation check box to be selected also. However, you can deselect this check box if desired.

For example, suppose you have the left-to-right screen that contains following text (capital letters are BIDI data and lower case letters remain as English data):

```
BIDI TEXT  pf01=help
```

You will customize this screen as RTL. When your application runs, the screen is displayed as:

```
TXET IDIB    pleh=10fp
```

From the end user point of view, everything is correct except the function key. The function key is recognized, but it is displayed backwards.

For the customized screen to appear correctly, you must select the **Widget Orientation opposite to Screen Orientation** check box when inserting the function key host component into a customized screen. By default, the **Text Orientation opposite to Screen Orientation** check box is also selected.

When your application runs, the screen is displayed as:

```
TXET IDIB    pf01=help
```

## Global variables

When global variables are extracted, the extracted data is exactly the data that appears on the screen, including the orientation of the current screen. Any global variables inserted onto a screen as an action of a screen customization are inserted according to the screen orientation.

## Text replacement

When you use text replacement in Bidi sessions, there are three additional check boxes you can use:

**Match with LTR Screens**
: This option allows text replacement to be performed correctly for text on a left-to-right display screen.

**Match with RTL Screens**
: This option allows text replacement to be performed correctly for text on a right-to-left display screen.

**Match with Reversed Screen**
: If you check this box and the **Match with LTR Screen** check box, the reversed string would match in a right-to-left screen. When **Reverse Screen** is clicked on a left-to-right screen, the data is consistent.

If you check this box and the **Match with RTL Screens** check box, the reversed string would match in a left-to-right screen. When **Reverse Screen** is clicked on a right-to-left screen, data is consistent.

You must check either **Match with LTR Screen** or **Match with RTL Screens**.

For example, suppose a left-to-right host screen contains the text: `NO on`. If you define text replacement to replace "no" with "yes" and ignore the case, the results depend on the boxes you checked, as follows:

*Table 3. Bi-directional text replacement options and results*

| Options selected | LTR screen | RTL screen |
|---|---|---|
| Match LTR screen only | yes on | no ON |
| Match RTL screen only | NO on | yes ON |
| Match LTR screen and RTL screen | yes on | yes ON |
| Match LTR screen and match reversed screen | yes on | no sey |
| Match RTL screen and match reversed screen | NO sey | yes ON |
| Match LTR and RTL screen and match reversed screen | yes sey | yes sey |

## Enabling the user to reverse the screen direction

You can provide a **Screen Reverse** button on bi-directional Web pages for the convenience of your end users. The button appears on the application keypad in the template. Clicking the button toggles the screen orientation—the operation performed on legacy systems by pressing the Alt+Enter keys.

When you create a new HATS project and select a bi-directional code page, two additional check boxes appear: **Enable screen reverse for uncustomized screens** and **Enable screen reverse for customized screens**. These check boxes determine whether the **Screen Reverse** button appears on these screens. Initially, only the first box is enabled. If you check the first box, the second box is enabled. There is no way to check only the second box.

An uncustomized screen is one that was not matched by any screen customization. An uncustomized screen, when viewed by the end user, has the same screen orientation as the previous screen. If the first screen is uncustomized, it defaults to left-to-right. If the screen orientation was changed on a previous screen, it is inherited by the next uncustomized screen and reset by the next customized screen.

The initial screen orientation of customized screens is the same as it was when the screens were customized. For both customized and uncustomized screens, clicking **Screen Reverse** changes the screen orientation. During screen recognition, a reversed screen is considered different from the same screen before the screen has been reversed. Therefore, **Screen Reverse** could cause a screen not to be recognized. If the developer is confident that all host components that appear on a customized screen are oriented properly, there is no need to enable the **Screen Reverse** button for that screen. It is advisable to disable the **Screen Reverse** button for customized screens.

When the user clicks **Screen Reverse**, the screen is refreshed and any entries the user has typed are erased.

## Information for end users

End users of HATS applications using bi-directional code pages can perform some operations unique to these applications. Provide the information in this section to your end users. If you implement a **Screen Reverse** button, inform them about its use.

HATS applications using bi-directional code pages offer a special input field called a visual input field. Unlike regular fields, which implement logical data input and presentation, the visual field implements visual data input and presentation. When entering data in a visual field, you can use these functions:

**Alt+Shift: Language selection**
This key combination toggles the language layer back and forth between Latin and the bi-directional language.

**Alt+Enter: Screen reverse**
This key combination reverses the direction of the screen.

**Shift+NumLock: Push**
You can enter and edit text whose direction is opposite from the field direction.

**Shift+NumPad: End push**
Push mode is ended and the cursor moves to the end of the push segment.

**Alt+NumPad: Auto push**
You can type mixed left-to-right and right-to-left text by changing the language layer. Autopush is especially useful for typing digits in right-to-left fields. The push and end push functions are automatically activated according to the language of the text being typed. In right-to-left fields, typing a digit or a Latin letter causes the automatic initiation of push, without a language change. Additional Latin letters or digits will continue the push mode; any other character automatically terminates push mode. This feature allows you to type bi-directional text with imbedded numbers or Latin words without using push and end push. In left-to-right fields, typing a bi-directional character causes the automatic initiation of push. Typing any digit or Latin character causes the automatic termination of the mode. This enables the end user to type Latin text with imbedded bi-directional words by using language layer selection rather than push and end push.

## Functions for Arabic code pages

These functions are specific to projects using Arabic code pages.

### Symmetric and numeric swapping

These options are effective only in Arabic 3270 sessions. If symmetric swapping is enabled, swapping characters are swapped in right-to-left screens. If numeric swapping 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. These parameters are set on the **Advanced Connection Settings** tab of the project editor. HATS host terminal is affected by symmetric swapping and numeric swapping parameters set by the user when creating the

application. The parameters are identified respectively as symmetricSwapEnabled and numericSwapEnabled, and the values of the parameters is either true or false.

### Screen captures

For an Arabic session with right-to-left captured screens, brackets and numerals are affected by the symmetric and numeric swapping options of the application. With WebSphere Studio Application Developer Version 4, to correctly view Arabic numbers the digit substitution should be set to “Contextual” in the regional settings.

### Other considerations

- When an end user enters data to be submitted to a HATS application, the shaping of Arabic data and Lam-Alef processing is performed according to the current screen orientation as the end user views it in the Web browser.
- To view Arabic numbers correctly in widget previews and in the deployed HATS application, digit substitution should be set to “None” in the regional settings.
- Screen recognition should always be done with whole Arabic words and not with a part of an Arabic word.

## Additions to HATS files

When a project uses an Arabic code page, screen customization event (.evnt) files have an additional <orientation>true</orientation> tag within the description tag. A value of true indicates that the screen is customized as a right-to-left screen; otherwise it is customized as left-to-right.

When a project uses any bi-directional code page, the application (.hap) file has an additional enableScrRev attribute of the <session> tag, that can have the following values:

**(blank)**
The **Screen Reverse** button is not placed on any screens.

**NotCustomized**
The **Screen Reverse** button is placed only on screens that do not match a screen customization.

**Customized**
The **Screen Reverse** button is placed on all screens.

When a project uses any bi-directional code page, the application (.hap) file has additional attributes for the <replace> tag within the <textReplacement> tag:

**matchLTR**
Text is to be replaced when the screen orientation is left-to-right.

**matchRTL**
Text is to be replaced when the screen orientation is right-to-left.

**matchReverse**
Text is to be replaced when the screen orientation is reversed.

## Bi-directional APIs

Two Bidi APIs for handling text conversion from visual to logical and vice versa are included in the HostScreen class, so you can use these APIs when creating custom widgets and components to handle 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.

**isLTRVisual**
If true, inputBuffer is in visual left-to-right form.

**isLTRimplicit**
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.

**isLTRimplicit**
If true, inputBuffer is in implicit left-to-right form.

**isLTRVisual**
If true, the output buffer is in visual left-to-right form.

# Appendix A. Component and widget descriptions and settings

HATS provides host components and widgets that are used to convert elements of a host screen to objects that can be displayed on a Web page. Some component and widget settings can be modified using the wizards and editors in the HATS Studio. This appendix describes HATS host components and widgets and the settings you can modify.

## Component and widget settings

The components and widgets supplied by HATS have default settings that you can modify, either for an entire project using the project editor, or for an individual transformation using the Insert Host Component wizard. Not all components and widgets have customizable settings.

### Host component settings

The settings for a host component specify how that component is to be recognized on the host screen. Some components' settings are very simple. For example, the only setting for a command line is the string of characters that identify a command line on the host screen. The default value is **===>**. If the command lines on your host screens are preceded by **==>**, they would not be recognized using the default setting. In this case, you would need to modify the setting so that the command lines would be recognized.

Some host components have more complicated settings. For example, several settings are used to recognize a function key or a selection list. These settings will be described under each host component.

HATS provides the following host components:

#### Command line

Consists of a string and an input field with the format:

==> [input field]

The ==> is called the token.

**Token** You can specify the token HATS uses to identify the command line. Type the value of the token that identifies the command line in the entry field.

#### Default

The contents of the selected region of the host screen. There are no customizable settings for the default component.

#### Field

A section of the host screen defined within a user-defined region of the host screen. There are no customizable settings for the field component.

#### Field table

A table in which each cell is a field that is defined on the host screen. Each field becomes a cell of the field table. HATS determines the table size based on the number of cells in a user-defined rectangular area of a 3270 or 5250 host screen. There are no customizable settings for the field table component.

## Function key

A horizontal list of function key names (F or PF) and host screen string descriptions with a delimiter between them (usually in the last row of the host screen). For example, the host screen could contain the following function keys:

**F3: Exit** **F4: Back** **F5: Fwd**

Function keys can have many different appearances on a host screen. To give you flexibility in recognizing function keys, HATS breaks down the appearance of a function key string into four parts. For example, a function key might look like this: `PF12=Exit`. In this example, the leading token (also known as the start delimiter) is `PF`, the delimiter (which separates the key number from the description) is `=`, and the description is `Exit`. There is no string before the leading token.

**String before the leading token**
: This value is optional. If there is a string that always precedes the start delimiter, such as ″option″, enter it here.

**Start delimiter**
: This is the string that marks the beginning of a function key string on the host screen. You can specify more than one value, separated by the ″|″ (vertical bar) character. Any of the values will be recognized as beginning a function key.

**Delimiter**
: This is the string that divides the function key number from its description. You can specify more than one value, separated by the ″|″ (vertical bar) character.

**String after the description**
: This string defines the end of the function key string on the host screen. It might be a blank character or the beginning of another function key string. You can specify more than one value, separated by the ″|″ (vertical bar) character.

## Input field

A field in which text can be entered, with or without the field label. There are no customizable settings for the input field component.

## Menu

Similar to the **Function key** host component; A menu is a list of choices, in which each choice is typically preceded by a letter or a number, with a delimiter character separating the letter or number from the text describing that choice. A menu choice might look like this: **option 12.Exit**. In this case, the word ″option″ is a string that precedes all the choices in the menu, and it is called the string before the leading token. The number 12 is the leading token, the period is the delimiter, and ″Exit″ is the description. Alternatively, you could have a menu whose choices look like this: **M: OPEN MAIL**, where the leading token is a letter, the delimiter is the colon, and ″OPEN MAIL″ is the description. There is no string before the leading token in this example.

**Delimiter**
: This is the string that divides the menu choice's leading token from its description. You can specify more than one value, separated by the ″|″ (vertical bar) character.

**String before the leading token**
This value is optional. If there is a string that always precedes the start delimiter, such as ″option″, enter it here.

**Minimum required options**
If you do not want to recognize a menu with fewer than a certain number of options, enter that number here.

## Selection list

A selection list is a lot like a menu, in that it presents a list of options, each of which is preceded by a leading token and a delimiter, such as in the following examples:

**1. Prepare form**
**2. Work with forms you submitted**
**3. Work with forms requiring action**
or
**a. Prepare form**
**b. Work with forms you submitted**
**c. Work with forms requiring action**

You can set the values of the following:

**Leading token type**
The leading token can be either a letter or a numeric digit.

**Delimiter**
This is the string that divides the selection's leading token from its description. In the examples, the delimiter is the period (.) following the numbers and letters. You can specify more than one value, separated by the ″|″ (vertical bar) character.

**String before the leading token**
This value is optional. If there is a string that always precedes the leading token, such as ″option″, enter it here.

**Minimum required options**
If you do not want to recognize a selection list with fewer than a certain number of options, enter that number here.

**Must be field separated**
Check this box if each selection must be in a separate field.

## Subfile

An iSeries or AS/400 screen with a pattern containing all of the following:

- A subfile fingerprint in the field attributes
- Subfile actions in the first half of the host screen
- Subfile headings in the first half of the host screen for the data that follows
- A subfile marker in the second half of the host screen (such as More...).
- Subfile data between the headings and marker, containing input fields or description text arranged in a table pattern.

There are no customizable settings for the subfile component.

## Text

Text that is located within a user-defined region of the host screen. There are no customizable settings for the text component.

### Visual table

A table based on the representation of text within a user-defined region of the host screen. By default, space breaks between text are interpreted as cell boundaries.

You can set the values of the following:

**Column delimiter**

For any character to be recognized as a column delimiter of a visual table, the character must be in the same column of each row in the selected region of the host screen. For example, a region is selected that includes rows 10 through 20. Assuming that a space is the column delimiter, and column 12 is assumed to identify a column, a space must be in column 12 of each row beginning at row 10 and continuing through and including row 20.

In the following example, a space is the column delimiter, and a space exists in column 7 of the host screen for the entire selected region of two rows and 10 columns:

```
rows     columns
        12345678910
 1      aaa cc ee
 2      bb ddd fff
```

For this selected region, HATS displays a 2x2 visual table with the following contents from the host screen:

| | |
|---|---|
| aaa cc | ee |
| bb ddd | fff |

Select from the drop-down list the string used to separate columns in the table.

**Include empty rows**

Check this box if you want empty rows in the visual table to appear in the HTML output. Clear this box if you want empty rows to be omitted.

**Rows to exclude**

Type the numbers of the rows in the table you want to exclude from the HTML presentation. If there is more than one row to exclude, separate the row numbers with a comma (,).

**Columns to exclude**

Type the numbers of the columns in the table you want to exclude from the HTML presentation. If there is more than one column to exclude, separate the row numbers with a comma (,).

## Widget settings

When you customize a host component, you are specifying how it will be recognized. When you customize a widget, you specify how the widget will appear on the Web page.

You can customize the settings of the following widgets:

### Button

Displays the host component as an HTML button. You can configure buttons to appear as a vertical or horizontal list by adjusting the number of columns in the display. For example, a HATS project could display buttons in one of the following configurations:

| | | |
|---|---|---|
| **[Prepare form]** | **[Work with forms you submitted]** | **[Work with forms requiring your action]** |

**[Prepare form]**
**[Work with forms you submitted]**
**[Work with forms requiring your action]**

The button widget presents a graphical representation of a link on the Web page, with a caption that describes its function. A button is created from a host component such as a function key or an item in a menu or a selection list. You can customize these settings for buttons:

**Number of columns per row**
Type the number of columns of buttons you want to display in each row.

**Caption type**
The values of the leading token and the description are derived from the host component. Choose whether you want the caption to display the leading token, the description, or both. For example, if the button represents a menu item that read **4.Mail**, you can have the caption display **4**, or **Mail**, or **4.Mail**.

**Caption substitution**
If you want to replace strings from the host component with new strings in the button caption, type the substitution values in the form of a=b. If you substitute more than one string, separate the substitutions with a semicolon (;).

## Button table

Displays a table of buttons created from host components such as menu items, where the first column contains buttons and the second column contains descriptive text. For example:

**[1]** Prepare form
**[2]** Work with forms you submitted
**[3]** Work with forms requiring your action

You can customize these settings for button tables:

**Number of columns per row**
Type the number of buttons you want to display in each row.

**Caption substitution**
If you want to replace strings from the host component with new strings in the button caption, type the substitution values in the form of a=b. If you substitute more than one string, separate the substitutions with a semicolon (;).

## Default

The default widget is used to represent an area of the host screen that might contain many different host components (the default component). The settings for the default widget contain information both about recognizing host components within the selected area of the screen and about how to present them.

The default widget includes numerous settings used to recognize and render PF keys from the host screen. You can specify two different ways of recognizing and rendering PF keys. These different ways are specified as the first and second

passes. HATS does not make two complete passes of the screen area; rather, HATS examines each character in the specified area, looking for PF key strings. If HATS encounters a character that matches the criteria defined by the first pass, HATS continues checking the characters that follow to determine whether the string fits the criteria to identify a PF key sequence of characters. If the first pass does not identify a PF key sequence, HATS returns to the first character that matched the first pass criteria. Each character is compared to the second pass criteria, to determine whether the string fits the criteria to identify a PF key sequence. If a string is recognized as a PF key sequence according to the first pass criteria, it is not checked against the second pass criteria.

**Preserve and map field colors**
Click the checkbox if you want to show the same colors for host fields in the HTML output.

**Use HTML Teletype (monospace) tag**
Click the checkbox if you want the HTML text output to be in a monospace font, which is close to the text spacing on the host screen.

**Perform 5250 subfile rendering**
Click the checkbox if you want 5250 subfile information rendered as subfiles in the HTML output.

**Perform selection list rendering**
Click the checkbox if you want selection list information automatically rendered in the HTML output. This is useful for applications (such as on iSeries) that use selections lists on many screens.

**Selection list widget**
Select one of the following widgets to use for rendering the selection list:

- Button
- Button table
- Dropdown list
- Link
- Option list.

**Leading token type**
The leading token can be a letter or a numeric digit.

**Delimiter**
This is the string that divides the selection's leading token from its description. In the examples, the delimiter is the period (.) following the numbers and letters. You can specify more than one value, separated by the "|" (vertical bar) character.

**String before the leading token**
This value is optional. If there is a string that always precedes the leading token, such as "option", enter it here.

**Minimum required options**
If you do not want to recognize a selection list with fewer than a certain number of options, enter that number here.

**First row to search for selection list**
Type the number of the first row on the host screen where a selection list might appear.

**Last row to search for selection list**
Type the number of the last row on the host screen where a selection list might appear.

**Number of columns per row**
Type the number of buttons, links, or options you want to display in each row. Unless the captions are short, one per row will look best.

**Caption type**
The values of the leading token and the description are derived from the host component. Choose whether you want each item of the widget selected to display the leading token, the description, or both. For example, if the item represents a menu item that read **4.Mail**, you can have the list item display **4**, or **Mail**, or **4.Mail**.

**Submit button caption**
Determines the text displayed on the submit button. You can select one of the values from the drop-down list (Submit, Go, or #=), or type the text you want the button to display in the entry field.

**Caption substitution**
If you want to replace strings from the host component with new strings on the button or link or in the drop-down or option list items, type the substitution values in the form of a=b. If you substitute more than one string, separate the substitutions with a semicolon (;).

**Show submit button**
Click the checkbox if you want to show a submit button in the HTML output. The user must click the button after choosing an option from the drop-down list or option list.

**Must be field separated**
Check this box if each selection must be in a separate field.

**Use character by character rendering for precise alignment**
Select the one of the following values from the drop-down list:

**Never** Character by character rendering is never used.

**Only on pages with popups**
Character by character rendering is only used on screens that have pop-up windows. The pop-up windows often look better with this alignment.

**Always**
Character by character rendering is always used.

**Convert menus to buttons**
Click the checkbox if you want to convert all menus on the host screen to buttons in the HTML output.

**First row to search for menus**
Type the number of the first row on the host screen where menus might appear.

**Last row to search for menus**
Type the number of the last row on the host screen where menus might appear.

**Substitute buttons or links for PF keys (first pass)**
Click the checkbox if you want to substitute buttons or links for PF keys identified by the criteria specified for the first pass.

**First pass HTML control for PF key**
You can replace PF keys with links or with buttons. On a link or a button, you can display the number of the key (such as PF12) or its description (such as ″Exit″). Choose one of these options from the drop-down list.

**First row to search for PF keys**
Type the number of the first row on the host screen to search for PF keys using the first-pass criteria.

**Last row to search for PF keys**
Type the number of the last row on the host screen to search for PF keys using the first-pass criteria.

**PF key substitution start delimiter**
Type the characters that define the beginning delimiter for a PF key for the first pass through the host screen. For `PF12=Exit`, `PF#` is the start delimiter.

**PF key substitution middle delimiter**
Type the characters that define the delimiter between a PF key and its description for the first pass through the host screen. For `PF12=Exit`, `=` is the middle delimiter.

**PF key substitution end delimiter**
Type the characters that define the ending delimiter for a PF key for the first pass through the host screen. The end delimiter could be something as simple as the space. However, if there are function key descriptions that contain a space, such as `PF8=Shift Right`, a space is not good to use as the end delimiter. In the example `PF8=Shift Right`, the end delimiter would have to be defined as the start of another function key, `PF#`, or more than a single space.

**Substitute buttons or links for PF keys (second pass)**
Click the checkbox if you want to substitute buttons or links for PF keys identified by the criteria specified for the second pass.

**Second pass HTML control for PF key**
You can replace PF keys with links or with buttons. On a link or a button, you can display the number of the key (such as PF12) or its description (such as ″Exit″). Choose one of these options from the drop-down list.

**First row to search for PF keys**
Type the number of the first row on the host screen to search for PF keys using the second-pass criteria.

**Last row to search for PF keys**
Type the number of the first row on the host screen to search for PF keys using the second-pass criteria.

**PF key substitution start delimiter**
Type the characters that define the beginning delimiter for a PF key for the second pass through the host screen. For `PF12=Exit`, `PF#` is the start delimiter.

**PF key substitution middle delimiter**
Type the characters that define the delimiter between a PF key and its description for the second pass through the host screen. For `PF12=Exit`, `=` is the middle delimiter.

**PF key substitution end delimiter**
Type the characters that define the ending delimiter for a PF key for the second pass through the host screen. The end delimiter could be something as simple as the space. However, if there are function key descriptions that contain a space, such as `PF8=Shift Right`, a space is not good to use as the end delimiter. In the example `PF8=Shift Right`, the end delimiter would have to be defined as the start of another function key, `PF#`, or more than a single space.

**Highlight tables**
Click the checkbox if you want to highlight the rows of the table, using two alternating colors, in the HTML output.

**Even tables row color**
Shows the background color to use for the even rows in the table. Click the button to display a color palette if you want to change the color.

**Odd tables row color**
Shows the background color to use for the odd rows in the table. Click the button to display a color palette if you want to change the color.

**Table title color**
Shows the color to use for the title of the table. Edit the entry if you want to change the color. The **Table title color** only applies if either the **First line of table is the title** or the **Line above table is the title** box is checked.

**Minimum row count in table**
If you do not want to recognize a table with fewer than a certain number of rows, enter that number here.

**Minimum column count in table**
If you do not want to recognize a table with fewer than a certain number of columns, enter that number here.

**First row to search for tables**
Type the number of the first row on the host screen where a table might appear on the host screen.

**Last row to search for tables**
Type the number of the last row on the host screen where a table might appear on the host screen.

**First line of table is the title**
Click the checkbox if you want the first line of the table to be used as the title of the table.

**Line above table is the title**
Click the checkbox if you want the line above the table to be used as the title of the table.

## Drop-down list

A drop-down list widget is a way of representing a large number of choices from a host menu or selection list without taking up a lot of room on the Web page. In the following example, HATS displays a drop-down list that shows three items.

**Work with forms**
**Work with forms you submitted**
**Work with forms requiring your action**

You can customize these settings for drop-down lists:

**Show submit button**
Click the checkbox if you want to show a submit button in the HTML output. The user must click the button after choosing an option from the list.

**Submit button caption**
Determines the text displayed on the submit button. You can select one of the values from the drop-down list (Submit, Go, or #=), or type the text you want the button to display in the entry field.

**Caption type**
The values of the leading token and the description are derived from the host component. Choose whether you want each item in the drop-down list to display the leading token, the description, or both. For example, if the item represents a menu item that read **4.Mail**, you can have the list item display **4**, or **Mail**, or **4.Mail**.

**Caption substitution**
If you want to replace strings from the host component with new strings in the drop-down list items, type the substitution values in the form of a=b. If you substitute more than one string, separate the substitutions with a semicolon (;).

## Field
Displays the host component in a field of the HATS project. There are no customizable settings for the field widget.

## Graph
Displays a graph in which the cells of a table (visual or field table) are divided into data sets.

You can set the values of the following:

**Number of data sets**
Determines the number of data sets to be included in the graph. The value defaults to the number of data sets found in the extracted data, but you can change the number to include only a subset of the data sets. This setting is only displayed in the settings for the Insert Host Component wizard, because it depends on the extracted data.

**Data set source**
Select one of the following values from the drop-down list:

**Row**
Each row of the table constitutes one set of data to be graphed.

**Column**
Each column of the table constitutes one set of data to be graphed.

**Width**
Type the width, in pixels, for the graph.

**Height**
Type the height, in pixels, for the graph.

**Background color**
Shows the color to use for the background in the graph. Click the button to display a color palette if you want to change the background color.

**Background image**
Type the path and name of the image to display in the background of the graph. Click the Browse button to locate an image on your system.

**X-axis title**
Type the text you want to use as the label for the X-axis in the graph.

**Y-axis title**
Type the text you want to use as the label for the Y-axis in the graph.

**Axis color**
Shows the color to use for the axis in the graph. Click the button to display a color palette if you want to change the axis color.

**Label color**
Shows the color to use for the label of the graph. Click the button to display a color palette if you want to change the label color.

As with the **Number of data sets**, the following are only displayed in the settings for the Insert Host Component wizard.

**Extract data point labels**
Click the checkbox if you want to extract row or column labels to show as labels on the X-axis.

**Row or Column**
Type in the entry field the number of the row or column to use as labels on the X-axis. The label for this entry field is dependent upon the value specified for the **Data set source** setting. The entry field label matches the value specified for the **Data set source** setting.

**Extract data set labels (for legend)**
Click the checkbox if you want to extract row or column labels to show as labels in the graph legend.

**'Row' or 'Column'**
Type in the entry field the number of the row or column of text to use as labels in the graph legend.

The label for this entry field is dependent upon the value specified for the **Data set source** setting. The entry field label is the opposite of the value for the **Data set source** setting.

**Data sets**
Click this button to display the Data Source Settings dialog, which enables you to specify the following additional settings for the data sources:

**Data set 'n', 'row' or 'column'**
The number (**'n'**) of these fields matches the value specified in the **Data set source** setting. Type in the entry field the number of any row or column of data you want to use for the data set. This enables you to reorder or duplicate sets of data in the graph.

The last part of the label for this entry field is dependent upon the value specified for the **Data set source** setting. The entry field label matches the value specified for the **Data set source** setting.

**color**
There is a color button for each of the **Data set 'n', 'row' or 'column'** settings. The buttons show the color to use for the data set in the graph. Click the button to display a color palette if you want to change the data set color.

## Label
Displays text for a labeled field on the host screen as a label for an input field in the HATS project. There are no customizable settings for the label widget.

## Link
The link widget presents a link on the Web page, with a caption that describes its function. A link is created from a host component such as a function key or an item in a menu or a selection list. You can configure links to appear as a vertical or horizontal list by adjusting the number of columns in the display. For example, a HATS project could display the links in one of the following configurations:

| Prepare form | Work with forms you submitted | Work with forms requiring your action |
|---|---|---|

**Prepare form**
**Work with forms you submitted**
**Work with forms requiring your action**

You can customize these settings for links:

**Number of columns per row**
Type the number of columns of links you want to display in each row. Unless the link captions are short, one link per row will look best.

**Caption type**
The values of the leading token and the description are derived from the host component. Choose whether you want the link caption to display the leading token, the description, or both. For example, if the link represents a menu item that read **4.Mail**, you can have the link caption display **4**, or **Mail**, or **4.Mail**.

**Caption substitution**
If you want to replace strings from the host component with new strings in the link caption, type the substitution values in the form of a=b. If you substitute more than one string, separate the substitutions with a semicolon (;).

## Option list

The option list widget is a way of presenting a list of mutually exclusive choices as radio buttons. For example:

**O** Prepare form
**O** Work with forms you submitted
**O** Work with forms requiring your action

You can customize these settings:

**Number of columns per row**
Type the number of options you want to display in each row.

**Show submit button**
Click the checkbox if you want to show a submit button in the HTML output. The user must click the button after choosing an option from the list.

**Submit button caption**
Determines the text displayed on the submit button. You can select one of the values from the drop-down list (Submit, Go, or #=), or type the text you want the button to display in the entry field.

**Caption type**
The values of the leading token and the description are derived from the host component. Choose whether you want each item in the drop-down list to display the leading token, the description, or both. For example, if the item represents a menu item that read **4.Mail**, you can have the list item display **4**, or **Mail**, or **4.Mail**.

**Caption substitution**
If you want to replace strings from the host component with new strings in

the drop-down list items, type the substitution values in the form of a=b. If you substitute more than one string, separate the substitutions with a semicolon (;).

### Table

Displays the selected information on the host screen as an HTML table.

You can set the values of the following:

**Header row**
Type the number of the row in the table on the host screen whose contents should be used as column headers on the Web page.

**Header column**
Type the number of the column in the table on the host screen whose contents should be used as row headers on the Web page.

**Read only**
Check this box if you want to prevent users from entering text into the table fields in the HTML output. Clear this box to enable users to enter text in the table cells.

**Row fill**
If a row does not have as many cells as other rows, it can be filled in by expanding the last cell to the end of the table (span) or by adding empty cells (empty). Choose one of these options from the drop-down list.

### Text input

Displays the selected information on the host screen as an input field, with or without the description. There are no customizable settings for the label widget.

## Component and widget mapping

The widgets supplied in HATS Studio for use in displaying host components on a Web page are mapped to those components. The following table lists the existing HATS host components and their corresponding widgets.

*Table 4. HATS host components and their corresponding widgets*

| Host component | Widget |
|---|---|
| Command line | Text input |
| Default | Default |
| Field | Field |
| Field table | Table<br>Horizontal bar graph<br>Vertical bar graph<br>Line graph |
| Function key | Button<br>Button table<br>Link<br>Option list |
| Input field | Text input |
| Menu | Button<br>Button table<br>Dropdown list<br>Link<br>Option list |

*Table 4. HATS host components and their corresponding widgets (continued)*

| Host component | Widget |
|---|---|
| Selection list | Button<br>Button table<br>Dropdown list<br>Link<br>Option list |
| Subfile | Subfile |
| Text | Label |
| Visual table | Table<br>Horizontal bar graph<br>Vertical bar graph<br>Line graph |

## HATS:Component tag type and widget attributes

The type and widget attributes of the HATS:Component tag are as follows:

**Attribute** **Description**

**type** The host component type. Valid values (and their corresponding host component) are:

| Type | Host component |
|---|---|
| **CommandLine** | Command line |
| **Default** | Entire host screen |
| **Field** | Field |
| **FieldTable** | Field Table |
| **FunctionKey** | Function key |
| **InputField** | Input field |
| **Menu** | Menu |
| **SelectionList** | Selection list |
| **Subfile** | Subfile |
| **Text** | Text |
| **VisualTable** | Visual table |

**widget** The widget style. Valid values (and their corresponding widgets) are:

| Widget | Widget |
|---|---|
| **Button** | Button |
| **ButtonTable** | Button table |
| **Default** | Entire host screen |
| **DropdownList** | Drop-down list |
| **Field** | Field |
| **HorizontalBarGraph** | Horizontal bar graph |
| **Label** | Label |

| | |
|---|---|
| **LineGraph** | Line graph |
| **Link** | Link |
| **OptionList** | Option list |
| **Subfile** | Subfile |
| **Table** | Table |
| **TextInput** | Text input |
| **VerticalBarGraph** | Vertical bar graph |

# Appendix B. HATS Studio files

When you use HATS Studio 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:** If you edit these source files, we recommend you use the HATS Studio editors.

All of the files you create with HATS Studio are stored on your system under the drive and directory where you installed your WebSphere Studio program, such as WebSphere Studio Application Developer. In the `workspace` subdirectory, a folder exists for each project with the name you supply when you create the project. For example, if you create a project and name it Employees, the files are stored in the following path:

*drive*`:/`*ws*`*`*d_dir*`/workspace/Employees`

where *drive* and *ws*d_dir* are the drive and directory where you installed the WebSphere Studio program.

All of the file locations in this appendix refer to the relative path from the directory named for your project.

## Application files (.hap)

The application file contains XML tags that define the settings you choose when you create the project.

The application (.hap) file is stored in the *project_name*`/source/profiles` directory, where *project_name* is the name you gave the project when you created it. You can view and edit the source of the application file by double-clicking on the **Project Settings** node of the **HATS Project View** to open the project editor. The source for the file can be viewed by clicking on the **Source** tab.

You can modify the application file using any of the tabs in the project editor. HATS Studio 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:

**description**
Specifies the description you enter when you create a project.

**template**
Specifies the name of the default template for the project, which you select when you create the project. The default template is Simple1.jsp.

## <sessions> tag

The **<sessions>** tag is the enclosing tag for the session characteristics.

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

**default**
Specifies the session configured for the project. This value should always be main, and main is the default.

## <session> tag

The **<session>** tag specifies the session characteristics for the project.

**Note:** If you select a bi-directional (BIDI) code page, refer to "Additions to HATS files" on page 86.

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

**codePage**
Specifies the numeric code page number for the codepage used in the project. The default value is 037. You select the codePage value when you create the project. A code page number might be used for more than one location or usage. See the description of the codePageKey attribute for the code page numbers.

**codePageKey**
Specifies the usage key that corresponds to the numeric codepage. The default value is KEY_US. Valid values for codePage and the location or usage key are:

*Table 5. Code pages and usage keys*

| Code page | Usage key |
|---|---|
| 037 | KEY_BELGIUM<br>KEY_BRAZIL<br>KEY_CANADA<br>KEY_NETHERLANDS<br>KEY_PORTUGAL<br>KEY_US |
| 273 | KEY_AUSTRIA<br>KEY_GERMANY |
| 274 | KEY_BELGIUM_OLD |
| 275 | KEY_BRAZIL_OLD |
| 277 | KEY_DENMARK<br>KEY_NORWAY |
| 278 | KEY_FINLAND<br>KEY_SWEDEN |
| 280 | KEY_ITALY |
| 284 | KEY_SPAIN<br>KEY_LATIN_AMERICA |
| 285 | KEY_UNITED_KINGDOM |
| 290 | KEY_JAPAN_KATAKANA_EX |
| 297 | KEY_FRANCE |
| 420 | KEY_ARABIC |
| 424 | KEY_HEBREW |

*Table 5. Code pages and usage keys (continued)*

| Code page | Usage key |
|---|---|
| 500 | KEY_MULTILINGUAL |
| 803 | KEY_HEBREW_OLD |
| 838 | KEY_THAI |
| 870 | KEY_BOSNIA_HERZEGOVINA<br>KEY_CROATIA<br>KEY_CZECH<br>KEY_HUNGARY<br>KEY_POLAND<br>KEY_ROMANIA<br>KEY_SLOVAKIA<br>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<br>KEY_BULGARIA<br>KEY_MACEDONIA<br>KEY_RUSSIA<br>KEY_SERBIA_MONTEGRO |
| 1026 | KEY_TURKEY |
| 1047 | KEY_OPEN_EDITION |
| 1112 | KEY_LATVIA<br>KEY_LITHUANIA |
| 1122 | KEY_ESTONIA |
| 1123 | KEY_UKRAINE |
| 1137 | KEY_HINDI |
| 1140 | KEY_BELGIUM_EURO<br>KEY_BRAZIL_EURO<br>KEY_CANADA_EURO<br>KEY_NETHERLANDS_EURO<br>KEY_PORTUGAL_EURO<br>KEY_US_EURO |
| 1141 | KEY_AUSTRIA_EURO<br>KEY_GERMANY_EURO |
| 1142 | KEY_DENMARK_EURO<br>KEY_NORWAY_EURO |
| 1143 | KEY_FINLAND_EURO<br>KEY_SWEDEN_EURO |
| 1144 | KEY_ITALY_EURO |
| 1145 | KEY_LATIN_AMERICA_EURO<br>KEY_SPAIN_EURO |
| 1146 | KEY_UNITED_KINGDOM_EURO |

*Table 5. Code pages and usage keys (continued)*

| Code page | Usage key |
|---|---|
| 1147 | KEY_FRANCE_EURO |
| 1148 | KEY_MULTILINGUAL_EURO |
| 1149 | KEY_ICELAND_EURO |
| 1153 | KEY_BOSNIA_HERZEGOVINA_EURO<br>KEY_CROATIA_EURO<br>KEY_CZECH_EURO<br>KEY_HUNGARY_EURO<br>KEY_POLAND_EURO<br>KEY_ROMANIA_EURO<br>KEY_SLOVAKIA_EURO<br>KEY_SLOVENIA_EURO |
| 1154 | KEY_BELARUS_EURO<br>KEY_BULGARIA_EURO<br>KEY_MACEDONIA_EURO<br>KEY_RUSSIA_EURO<br>KEY_SERBIA_MONTEGRO_EURO |
| 1155 | KEY_TURKEY_EURO |
| 1156 | KEY_LATVIA_EURO<br>KEY_LITHUANIA_EURO |
| 1157 | KEY_ESTONIA_EURO |
| 1158 | KEY_UKRAINE_EURO |
| 1160 | KEY_THAI_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 |

**delayInterval**
Specifies the time (in milliseconds) that the server waits until a full host screen that is not the first host screen has arrived. The initial default value is 1500 milliseconds.

**delayStart**
Specifies the time (in milliseconds) that the server waits until the first full host screen has arrived. The initial default value is 1500 milliseconds.

**description**
Specifies a description for the session configured for the project. This value is always empty.

**enableSSL**
Specifies whether SSL is enabled. Valid values are:

**true** SSL is enabled for the project.

**false** SSL is not enabled for the project.

**enhanced**
Specifies whether the connection is a TN3270E connection. Valid values are true and false. The initial default is true.

**hostName**
Specifies the name of the host to which the project connects.

**name** Specifies the session configured for the project. This value should always be main, and main is the initial default.

**port** Specifies the number of the port through which the connection to the host is made. The initial default is 23.

**printFontName**
Specifies the font in which you want your output printed. Valid values depend on the value of the codePage attribute.

**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

**printSupport**
Specifies whether your project includes print capability. Valid values for printSupport are true and false. The initial default is false.

**printURL**
Specifies the URL for an iSeries for Web Access (IWA) Printer Output window on a 5250 server. The default URL is http://*hostname*/webaccess/iWASpool, where *hostname* is the name of the 5250 server.

**screenSize**
Specifies the number of rows and columns that the host terminal displays. Valid values for screenSize are:

- 24 x 80
- 27 x 132
- 32 x 80 (3270 only)
- 43 x 80 (3270 only)

The initial default screen size is 24 x 80.

**type** Specifies the type of terminal the host terminal displays. Valid values for type are:

- 3270
- 3270E
- 5250

The initial default is 3270.

## <otherParameters> tag

The **<otherParameters>** tag specifies additional Host On-Demand session parameters.

Host On-Demand session parameters supported by HATS are:

**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 is LAMALEF_OFF .

**LUName**
Sets the LUName property, which is the LU name used during enhanced

negotiation. This property is only valid when the TNEnhanced property is true. This property is valid for 3270 sessions only. Values are in string format. Maximum length of LUName is 17 characters. There is no default.

**numeralShape**
Sets the numeralShape property. This property applies to bi-directional sessions only. Values are in string format. The default is NOMINAL.

**numericSwapEnabled**
Sets the Numeric swapping property. This property applies to Arabic 3270 sessions only. Valid values are true and false. The default is true.

**roundTrip**
Sets the roundTrip property. This property applies to bi-directional sessions only. Values are in string format. Valid values are:

- ROUNDTRIP_ON
- ROUNDTRIP_OFF

The default 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 is TLS.

**SSLServerAuthentication**
Sets the SSLServerAuthentication property, which indicates whether SSL server authentication is enabled. Valid values are true and false. The default is false.

**symmetricSwapEnabled**
Sets the Symetric swapping property. This property applies to Arabic 3270 sessions only. Valid values are true and false. The default is true.

**textOrientation**
Sets the textOrientation property. This property applies to bi-directional sessions only. Values are in string format. Valid values are:

- LEFT_TO_RIGHT
- RIGHT_TO_LEFT

The default is LEFT_TO_RIGHT.

**ThaiDisplayMode**
Sets Thai display mode property. This property applies to Thai sessions only. Values are in string format. The default 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 will be converted to uppercase. There is no default.

## <eventPriority> tag

The **<eventPriority>** tag is the enclosing tag for the screen customization events you defined for the project. The order of the event tags within the **<eventPriority>** tag defines which events have higher priority. The highest priority event should be the first event in the list.

## <event> tag

The **<event>** tag specifies an event you defined for the project.

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

**enabled**
Specifies whether the event can occur within the project. Valid values for enabled are true and false. The default is true.

**name** Specifies the name you gave the screen customization event when you defined it. If you store a screen customization file under a folder (or group), the name of the folder is prepended to the name of the file.

## <classSettings> tag

The **<classSettings>**tag is the enclosing tag for the Java classes you include in the project.

## <class> tag

The **<class>**tag specifies the Java classes that can be included in an project.

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

**name** Specifies one of the following Java classes:

- com.ibm.hats.common.ApplicationKeypadTag
- com.ibm.hats.common.HostKeypadTag
- com.ibm.hats.common.KeyboardSupport
- com.ibm.hats.component.*

  where * is the name of a component for which you have customized a setting
- com.ibm.hats.widget.*

  where * is the name of a widget for which you have customized a setting.
- com.ibm.hats.common.ClientLocale

The class names on the name attribute must be enclosed in quotes.

## <setting> tag

The **<setting>**tag specifies the methods included in the Java class.

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

**name** Specifies the name of the Java method or a customized setting for a component, a widget or the locale. The names listed depend on the Java class in which the methods reside or the name of a component or widget setting.

For the com.ibm.hats.common.ApplicationKeypadTag class, the methods are:

**show** If value=true, shows a keypad in the application.

**style** Depending on the value attribute, shows the keys defined with value=true as either a button or a link in the application keypad.

**showKeyboardToggle**
If value=true, shows a key in the application keypad for toggling display of a host keyboard.

**showPrintJobs**
If value=true, shows a key in the application keypad for showing print jobs.

**showReset**
If value=true, shows a Reset key in the application keypad to clear all the fields on the browser page of any entries made by the end user.

**showReverse**
If value=true, shows a Reverse key in the application keypad for bi-directional support.

**showRefresh**
If value=true, shows a Refresh key in the application keypad to refresh the browser window contents using the original transformation, and restore the input fields to their original value.

**showDisconnect**
If value=true, shows a Disconnect key in the application keypad to disconnect from the host.

**showDefault**
If value=true, shows a Default key in the application keypad to change the presentation to the default transformation.

For the com.ibm.hats.common.HostKeypadTag class, the methods are:

**show** If value=true, shows a host keypad in the application.

**style** Depending on the value attribute, shows the keys defined with value=true as either a button or a link in the host keypad.

**showAttention**
If value=true, shows an ATTN key in the host keypad.

**showPrint**
If value=true, shows a PRINT key in the host keypad for printing output.

**showSystemRequest**
If value=true, shows a SYSREQ key in the host keypad.

**showClear**
If value=true, shows a CLEAR key in the host keypad.

**showPageUp**
If value=true, shows a Page Up key in the host keypad.

**showPageDown**
If value=true, shows a Page Down key in the host keypad.

**showPA1**
If value=true, shows a PA1 key in the host keypad.

**showPA2**
If value=true, shows a PA2 key in the host keypad.

**showPA3**
If value=true, shows a PA3 key in the host keypad.

**showEnter**
If value=true, shows an Enter key in the host keypad.

**showAltView**
If value=true, shows an AltView key in the host keypad.

**showHelp**
If value=true, shows a Help key in the host keypad.

**showF1 – showF24**
If value=true, shows a Function key with a number in the host keypad.

For the com.ibm.hats.common.KeyboardSupport class, the methods are:

**enable**
Depending on the value attribute, enable specifies whether keyboard support is available in the project.

**initialState**
If value=true, the initial state of the host keyboard is on (the physical keys on the keyboard are active).

For the com.ibm.hats.component.* class or com.ibm.hats.widget.* class, name specifies a customized component or widget setting.

For the com.ibm.hats.common.ClientLocale class, name is always `locale`.

**value** For definitions of keypad keys, specifies whether to show the key in the keypad. Valid values are true and false.

For name=style, specifies how keys defined with value=true are displayed in the in the host keypad. Valid values are the following:

- Buttons
- Links

For component or widget settings, value specifies what you specified for the customized setting.

For the com.ibm.hats.common.ClientLocale class, value specifies characters that identify the country code of the `locale`.

## <textReplacement> tag

The **<textReplacement>** tag is the enclosing tag for any text replacement values you define in the project.

## <replace> tag

The **<replace>** tag specifies the text replacement values in a project.

**Note:** If you are using a bi-directional (BIDI) code page, refer to "Additions to HATS files" on page 86.

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.

**from** Specifies the text you want to replace. The text on the from attribute must be enclosed in quotes.

**to** Specifies the text you want to insert in place of the value specified on the from attribute. The text on the to attribute must be enclosed in quotes.

**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 could be contracted, expanded, or forced to a new line.

## 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*/webApplication/templates directory. The transformation .jsp files are stored in the *project_name*/webApplication/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 the **Design** or the **Source** tabs in the JSP editor. HATS Studio updates the affected information on other tab when you make changes on either tab.

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 "Creating custom host components and widgets" on page 55.

## 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*/source/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**, or **Source** tabs in the editor. HATS Studio 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 to occur when the host screen is recognized. The tags are:

**event** 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.

**actions**
Encompasses the actions defined in the screen customization. The possible actions and their attributes are:

**apply** Defines the action for applying a transformation. The attributes of the apply tag are:

**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** Defines the action for inserting a global variable or a string. The attributes of the insert tag are:

**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.

**extract** Defines the action for extracting a global variable. The attributes of the extract tag are:

**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.

**set** Defines the action for setting a global variable. The attributes of the set tag are:

**name** Specifies the name of the global variable being set. This can be an existing global variable or a new global variable.

**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.

**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.

**op** Specifies the type of operation to occur between the first and second operands of a calculated value. Valid values are + (add), - (subtract), * (multiply), / (divide), and % (percentage).

**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 first 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.

**dec** Specifies the number of decimal places to which a calculated value is rounded. Valid values are 0–999.

**execute**
Defines the action for executing business logic. The attributes of the execute tag are:

**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** Defines the action for showing a URL. The show tag has the following attribute:

**url** Identifies the Uniform Resource Locator (URL) of the Web page to show. This attribute is required.

**play** Defines the action for playing a macro. The play tag has the following attribute:

**macro** Names the macro to be played. This attribute is required.

**associatedScreens**
The associatedScreens tag encompasses the screen tag that follows.

**screen** 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**
The description tag is the enclosing tag for the description associated with the screen customization, which is comprised of the oia tag and the string tag. There are no attributes for the description tag.

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

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

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

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

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

## Macro files (.hma)

Macro files are stored in the *project_name*/source/profiles/events/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.

You can modify macro files using the **Prompts and Extracts** or **Source** tabs in the editor. HATS Studio updates the affected information on the other tab when you make changes on either tab.

Macro files contains tags that define a set of screens. The tags are:

**macro** Begins the definition of the macro. The macro tag has no attributes.

**extracts**
The associatedScreens tag encompasses the extract tag that follows. The extracts tag has no attributes.

**extract** 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 Studio 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.

**showHandler**
Specifies whether the extracted information should be shown to the end user. Valid values are true and false.

**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.

**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.

**prompts**
The prompts tag encompasses the prompt tag that follows. The prompts tag has no attributes.

**prompt**
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 Studio 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.

**value** Specifies either the string to be used for the prompt or the name of a global variable from which the value is taken.

**HAScript**
The HAScript tag is the main enclosing tag for the other macro tags and attributes.

For descriptions of the HAScript tag, its attributes, and other tags used with the HAScript tag in macros, see Appendix C, "Macro script syntax" on page 123.

## Screen capture files (.hsc)

Screen capture files are XML representations of host screens, used to create or customize screen customizations or transformations.

Screen capture files are stored in the *project_name*/screens 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.

## Image files (.gif or .jpg)

Image files are used in HATS Studio within template files to create the Web page displayed to the user of your project.

Image files are stored in the *project_name*/webApplication/common/images directory. You can view and edit the image files by double-clicking on the name of the image in the **HATS Project View** to open the WebSphere Studio WebArt Designer.

## Stylesheet files (.css)

Stylesheet files are used in HATS Studio within template files to specify appearance items such as color, font, font size, whitespace, and spacing between letters.

Stylesheet files are stored in the *project_name*/webApplication/common/stylesheets directory. You can edit the stylesheet files by double-clicking on the name of the stylesheet in the **HATS Project View** to open the stylesheet editor.

# Appendix C. Macro script syntax

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

## Introduction

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

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

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

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

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

```
<HAScript>
  <screen name="Logon" entryscreen="true">
     <description>
         <string value="Please Logon" casesense="true"/>
         <cursor row="12" col="10"/>
     </description>
     <actions>
         <prompt name="ID" row="12" col="10" len="8"/>
         <prompt name="Password" row="13" col="10" len="8"/>
         <input value="[enter]"/>
     </actions>
     <nextscreens>
          <nextscreen name="Logon.Complete"/>
     </nextscreens>
  </screen>
<HAScript>
```

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

Keep in mind that the actions happen in sequence. The **<screen>** tag describes a logon screen with the text `Please Logon` on the screen and the screen's cursor position at row 12, column 10. If the macro logic sees a screen matching this description, it prompts the user for an ID and password, places the prompt results at the specified row and column positions, and sends the ENTER key, effectively logging on the user. The **<nextscreens>** tag specifies a list of **<nextscreen>** tags, and the **<nextscreen>** tags list the names of other **<screen>** tags that appear later in the macro. If a next screen does not appear, the macro logic returns an error.

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

## Macro

The following are valid macro tags:

```
<HAScript>
  <vars>
    <create>
  <screen>
    <comment>
    <description>
      <oia>
      <cursor>
      <numfields>
      <numinputfields>
      <string>
      <attrib>
      <customreco>
      <varupdate>
  <actions>
      <prompt>
      <input>
      <extract>
      <message>
      <trace>
      <filexfer>
      <pause>
      <mouseclick>
      <boxselection>
      <commwait>
      <custom>
      <varupdate>
      <playmacro>
      <if>
      <else>
      <runprogram>
  <nextscreens>
      <nextscreen>
  <recolimit>
```

These XML tags and their attributes are valid in the IBM Host On-Demand Macro XML namespace. This description of the tags is structured like an actual macro file. The tag and attribute values are not case sensitive.

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

## <HAScript> tag

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

**Note:** You cannot use variables as the values for HAScript tag attributes.

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

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

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

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

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

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

**pausetime**
The sleep time in milliseconds initiated after a screen is matched. This is used to let the host quiet down. This attribute is optional. The value must be a number. The default is 300 milliseconds. If a <pause> tag is specified for a specific screen, the value specified on the <pause> tag overrides this value.

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

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

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

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

recognition event to be processed and the valid blank screen will match when it shouldn't have matched. This attribute is optional. The default is false. The value must be true or false.

**usevars**

Determines whether attribute values are interpreted as literal values or as variables. This attribute is optional. Its value must be true or false. The default is false; however, if a variable is created using the Host On-Demand Macro Editor, the value of usevars is automatically set to true.

If you plan to use variables in your macro (including inherited variables from a parent macro), set this attribute to true. The values of macro element attributes are then parsed for variable names and arithmetic operators. (For example, $var$ would be interpreted as a variable.) See "Using variables" on page 147 for instructions on how to use the reserved characters single quote (') and backslash (\).

If you do not plan to use variables in your macro, either set this attribute to false or leave it unset.. The values of macro element attributes are interpreted as literal strings or numeric values (as appropriate), including the names of variables and arithmetic operators.(For example, $var$ would be interpreted as the literal string "$var$"; the dollar sign ($) and the arithmetic operators (+, -, *, and /) would also be interpreted as string characters.) Leaving the usevars attribute set to false allows macros created under Host On-Demand Version 6 and earlier to run under Host On-Demand Version 7 without modification, since variables were not supported for these releases.

If usevars is set to false and the macro parser finds a <vars> tag in the macro, the parser displays an error message telling you to set usevars to true.

### Example

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

## <vars> tag

Defines variables that are used in the macro if the usevars attribute of the <HAScript> tag is set to true. (If usevars is set to false and the macro parser finds a <vars> tag in the macro, the parser displays an error message telling you to set usevars to true.

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

Including the <vars> tag in a macro has the same effect as defining variables through the **Variables** tab in the Host On-Demand Macro Editor. The <vars> tag must occur before the <screen> tag in a macro.

Use the <create> tag to declare variables and assign initial values to them. See "Using variables" on page 147 for detailed information on how variables can be used in macros.

Variables can be inherited from another macro. See the description of the <playmacro> tag for details.

### Example

```
<HAScript usevars="true">
  <vars>
      ... #Variable declarations
  </vars>
</HAScript>
```

## <create> tag

Declares variables and assigns initial values to them. See "Using variables" on page 147 for detailed information on how variables can be used in macros.

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

**name** The unique name of the variable. Variable names are specified in the following format: $*var_name*$, where *var_name* can contain alphanumeric characters, the dash character (-), and the underscore character (_). Variable names are case sensitive.

**type** The data type of the variable. The following data types are supported:

**boolean**
Represents true or false boolean values.

**integer**
Represents integer numbers.

**double**
Represents double-precision numbers.

**string** Represents text strings.

**field** Represents text in a field on the terminal screen. The position within the field is given as `row,column`. (For example, 2,3 represents the second row, third column of the screen.)

**value** The inital value that is assigned to the variable. This attribute is optional. Initial values can be assigned to the different variable types as follows:

**boolean**
A boolean variable can be assigned a value of either "true" or "false". Note that boolean variables are not case sensitive: values such as "False" or "FALSE" are valid. A string variable or a field variable can be assigned to a boolean variable, as long as the string or field variable contains a valid boolean value. The default value is false.

**integer**
An integer variable can be assigned an actual integer value (such as 15 or -2) or the results of an arithmetic operation. The default value is 0.

**double**
A double variable can be assigned an actual double-precision value or the results of an arithmetic operation. The default value is 0.0.

**string** A string variable can be assigned combinations of boolean, integer, double, string, or field variables, and actual text strings. The default value is "", (an empty string).

**field** No initial value can be assigned to a field variable. The default value is an empty field.

### Example

```
<vars>
  <create name="$var_boolean$" type="boolean" value="true"/>
  <create name="$var_int$" type="integer" value="1"/>
  <create name="$var_double$" type="double" value="1.0"/>
  <create name="$var_string$" type="string" value="some_texts"/>
  <create name="$var_field$" type="field" value="field"/>
</vars>
```

## <screen> tag

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

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

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

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

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

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

**transient**
If true, the screen is handled as transient. Transient screens exist outside the normal macro flow. **They are matched after nontransient screens. If you specify next screens in a transient screen, the next screens are ignored**. Use this attribute to specify a throw-away screen that can appear at any time in the screen flow. This value must be true or false or an expression that evaluates to true or false. This attribute is optional. The default is false.

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

### Example

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

## <comment> tag

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

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

### Example

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

## <description> tag

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

By default, when the Macro Manager records a macro, the OIA and Field Counts descriptors are defined to identify the screen. It is recommended that you add String descriptors for more strict and accurate screen recognition.

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

**uselogic**
Determines the boolean logic for screen recognition. The numbers in the value represent the sequential positions of the descriptors in the **<description>** tag. There must be a descriptor for each number in the value. See "Advanced Screen Recognition" on page 146 for detailed information on using the uselogic attribute.

### Example

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

## <oia> tag

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

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

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

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

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

**invertmatch**
If true, recognition matching passes only if the screen does not match this description element (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.

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

### Example

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

## <cursor> tag

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

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

**row** The row position of the cursor. The value must be a number or an expression that evaluates to a number. This is a required attribute.

**col** The column position of the cursor. The value must be a number or an expression that evaluates to a number. This is a required attribute.

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

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

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

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

### Example

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

## <numfields> tag

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

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

**number**
The field count. The value must be a number or an expression that evaluates to a number. This is a required attribute.

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

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

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

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

### Example

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

## <numinputfields> tag

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

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

**number**
The field count. The value must be a number or an expression that evaluates to a number. This is a required attribute.

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

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

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

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

### Example

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

## <string> tag

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

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

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

**row** The starting row position for a string at an absolute position or in a rectangle. The value must be a number 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 are comprised of more than one non-optional descriptor, and more than one optional descriptor, the non-optional descriptors are checked first. If all of the non-optional descriptors match, the screen matches. If at least one of the non-optional descriptors does not match, the optional descriptors are checked. One of the optional descriptors must match for the screen to match. Otherwise, the screen fails to match. The value must be true or false or an expression that evaluates to true or false. This attribute is optional. The default is false.

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

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

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

## Examples

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

## <attrib> tag

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

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

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

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

**row** The row position of the attribute. The value must be a number or an expression that evaluates to a number. This is a required attribute.

**col** The column position of the attribute. The value must be a number or an expression that evaluates to a number. This is a required attribute.

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

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

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

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

### Example

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

## <customreco> tag

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

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

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

**optional**
If false, this descriptor is considered non-optional during screen

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

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

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

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

### Example

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

## <varupdate> tag

Modifies the values of variables. This element may be used anywhere within the <description> and <actions> tag blocks. For more information about macro variables, see "Using variables" on page 147.

By default, <varupdate> commands are the first commands to be executed in the <description> section - regardless of where they occur sequentially in the section. For example, if you create a String descriptor and use a variable for the string, then update the variable right after that descriptor, the variable is updated before the String descriptor is checked for a match. You can change the order of execution of <varupdate> tags by using the <description> tag's uselogic attribute; see "Advanced Screen Recognition" on page 146 for details.

**Note:** When a <varupdate> tag is used in a <description> block, the variable's value is updated when the macro attempts to match that screen, not when the screen is actually matched. This means that the variable is updated even if the screen doesn't match.

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

**name** The unique name of the variable. Variable names are specified in the following format: $*var_name*$, where *var_name* is the name assigned to the variable using the <create> tag.

**value** The new value to be assigned to the variable. This attribute is optional; updating with a value of "" (null) resets the variable to its default value (boolean to false, integer to 0, and so forth). Assign values to the different variable types as follows:

**boolean**
A boolean variable can be assigned a value of either "true" or "false". Note that boolean variables are not case sensitive: values such as "False" or "FALSE" are valid. A string variable or a field

variable can be assigned to a boolean variable, as long as the string or field variable contains a valid boolean value.

**integer**
An integer variable can be assigned an actual integer value or the results of an arithmetic operation. An integer variable must be assigned a valid integer number (for example, 3 or -4). If it is updated with a non-integer value, the decimal portion of the value is truncated (for example, if an integer variable is assigned a value of 4.8, the assigned value is truncated to 4).

**double**
A double variable can be assigned an actual double-precision value or the results of an arithmetic operation when it is created.

**string** A string variable can be assigned combinations of boolean, integer, double, string, or field variables, and actual text strings. Literal strings must be enclosed in single quotation marks (').

**field** A field position consists of two integers separated by a comma (for example, 2,3). After a field position value is provided by the <varupdate> tag, the text of the field containing the specified field position is assigned to the field variable at run-time. If the value given for the field position does not evaluate to "integer, integer" at runtime, a runtime error occurs unless the field's value has been set to null (""). If a field variable is used before a value is assigned to it with the <varupdate> tag, its value is automatically set to null ("").

### Example

```
<screen>
  <description>
    <varupdate name="$var_boolean$" value="false"/>
    <varupdate name="$var_int$" value="5"/>
    <varupdate name="$var_double$" value="5"/>
    <varupdate name="$var_string$" value="new_texts"/>
    <varupdate name="$var_field$" value="4,5"/>
  </description>
</screen>
```

## <actions> tag

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

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

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

### Example

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

## <prompt> tag

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

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

**row** The row to place the prompt. The value must be a number or an expression that evaluates to a number. This is a required attribute.

**col** The column to place the prompt. The value must be a number or an expression that evaluates to a number. This is a required attribute.

**len** The length of the prompt. The value must be a number or an expression that evaluates to a number. This is a required attribute.

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

**description**
The text that is displayed as the prompt. This can be any valid Unicode character. The description text must be enclosed in single quotes (’). This attribute is optional.

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

**clearfield**
This clears the host field on placement of prompt text. The value must be true or false or an expression that evaluates to true or false. This attribute is optional. The default is false.

**encrypted**
Use a password echo character. The value must be true or false or an expression that evaluates to true or false. This attribute is optional. The default is false.

**xlatehostkeys**
If true, host key mnemonics (example, [enter]) will be translated. For a list of key mnemonics, see the Host On-Demand online help. The value must be true or false or an expression that evaluates to true or false. This attribute is optional. The default is false. If you do not have this value set to true (which is normal because you wouldn’t ask users to type key mnemonics), don’t forget to code an input tag after the prompt(s) for the current actions to get the prompt data entered onto the host.

**assigntovar**
Assigns the user’s input to a variable. The entered value must be of the same data type as the variable; otherwise, an error or an unexpected result occurs. This attribute is optional.

**varupdateonly**
If true, assigns the user’s input to a variable without displaying the value to the screen. The default is false. This attribute is optional.

### Example

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

## <input> tag

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

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

**row** The row position to send the keys. The value must be a number or an expression that evaluates to a number. This attribute is optional. This defaults to current cursor position.

**col** The column position to send the keys. The value must be a number or an expression that evaluates to a number. This attribute is optional. This defaults to current cursor position.

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

**value** The text that is sent to the screen. This can be any valid Unicode character. The text must be enclosed in single quotes (’). This is a required attribute.

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

### Example

```
<input value="IBM[tab] is cool [enter]" row="1" col="1" movecursor="true"
          xlatehostkeys="true" />
```

## <extract> tag

The **<extract>** tag specifies an area where the screen’s contents are to be extracted.

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

**name** The name of the extract. This can be any valid Unicode character. The name must be enclosed in single quotes (’). This attribute is optional.

**srow** Upper left row of the bounding extract rectangle. The value must be a number or an expression that evaluates to a number. This is a required attribute.

**scol** The upper left column of the bounding extract rectangle. The value must be a number or an expression that evaluates to a number. This is a required attribute.

**erow** The lower right row of the bounding extract rectangle. The value must be a number or an expression that evaluates to a number. This is a required attribute.

**ecol** The lower right column of the bounding extract rectangle. The value must be a number or an expression that evaluates to a number. This is a required attribute.

**unwrap**
If this value is set to true, the macro bean will use the underlying screen fields to unwrap text that is in a field that spans multiple lines on the screen. This will result in an extract String array that has less elements than the row and col values would indicate. The value must be true or false, or an expression that evaluates to true or false. The default is false.

**assigntovar**
Assigns the extracted characters to a variable. If multiple rows of text are extracted, the variable contains values as one row concatenated to the end

of the next (second row appends to end of first row, third row appends to second row, and so forth), like the following:

```
Row1textRow2textRow3textRow4text
```

The last row does not contain a new line character (/n). The variable can be of any data type. If the variable is an integer or a double, the extracted text must be of a matching data type; otherwise an error or unexpected result occurs.

### Example

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

## <message> tag

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

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

**title** The title to display in the message dialog. This can be any valid Unicode character. The message title must be enclosed in single quotes (’). This attribute is optional. This defaults to macro name.

**value** The message to display in the dialog. This can be any valid Unicode character. The message must be enclosed in single quotes (’). This is a required attribute.

### Example

```
<message value="Accessing Host System" title="Message from IBM" />
```

## <trace> tag

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

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

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

**value** The text that is sent to trace. This can be any valid Unicode character. Trace value text must be enclosed in single quotes (’). This is a required attribute.

### Example

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

## <filexfer> tag

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

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

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

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

**Note:** To include a backslash (\) in a file name, you must specify two backslashes (\\) in the macro; see "Using Special Characters in Macros" on page 157 for details.

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

**clear** Indicates whether the macro bean should clear the host screen before performing the file transfer. The value must be true or false or an expression that evaluates to true or false. This attribute is optional.

**timeout**
Sets the time out value (in milliseconds) for the file transfer. If the transfer does not complete in this given time, the macro will end in error. The value must be a number (or an expression that evaluates to a number) in milliseconds. This attribute is optional and the default is 10000 milliseconds or 10 seconds.

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

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

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

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

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

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

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

### Example

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

## <pause> tag

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

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

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

### Example

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

## <mouseclick> tag

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

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

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

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

### Example

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

## <boxselection> tag

The **<boxselection>** tag is used for either marking or unmarking the marking rectangle on the terminal bean.

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

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

**scol** The upper left column of the bounding selection rectangle. The value must be a number (or an expression that evaluates to a number) within the host screen coordinate system. Negative values are allowed and specify a virtual position from the last column. This is a required attribute.

**erow** The lower right row of the bounding selection rectangle. The value must be a number (or an expression that evaluates to a number) within the host

screen coordinate system. Negative values are allowed and specify a virtual position from the last row. This is a required attribute.

**ecol** The lower right column of the bounding selection rectangle. The value must be a number (or an expression that evaluates to a number) within the host screen coordinate system. Negative values are allowed and specify a virtual position from the last column. This is a required attribute.

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

### Example

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

## <commwait> tag

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

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

**value** The type of communication status to wait for. Valid values are CONNECTION_INIT, CONNECTION_PND_INACTIVE, CONNECTION_INACTIVE, CONNECTION_PND_ACTIVE, CONNECTION_ACTIVE, CONNECTION_READY, and CONNECTION_DEVICE_NAME_READY. The meaning of these types is documented in the Java documentation for the ECLConnection object in the Host On-Demand Host Acceess Class Library documentation. The two most used and most meaningful types are CONNECTION_READY and CONNECTION_INACTIVE. This is a required attribute.

**timeout**
Sets the time out value (in milliseconds) for the communication wait. If the wait does not complete in this given time, the macro will end in error. The value must be a number (or an expression that evaluates to a number) in milliseconds. This attribute is optional and the default is no time out.

### Example

```
<commwait value="CONNECTION_READY" timeout="10000" />
```

## <custom> tag

The **<custom>** tag enables the user to have an exit to Java code. See the Host On-Demand Java documentation for the MacroActionCustom class.

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

**id** The ID of the callout code that the macro bean will use. This can be any valid Unicode character. This is a required attribute.

**args** The argument string that can be passed to the callout. This can be any valid Unicode character. String text must be enclosed in single quotes (’). This attribute is optional.

### Example

```
<custom id="custom1" args="IBM means world class computers" />
```

## <varupdate> tag

Modifies the values of variables. This element may be used anywhere within the <description> and <actions> tag blocks. For more information about macro variables, see "Using variables" on page 147.

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

**name** The unique name of the variable. Variable names are specified in the following format: $*var_name*$, where *var_name* is the name assigned to the variable using the <create> tag.

**value** The new value to be assigned to the variable. Assign values to the different variable types as follows:

**boolean**
A boolean variable can be assigned a value of either "true" or "false". Note that boolean variables are not case sensitive: values such as "False" or "FALSE" are valid. A string variable or a field variable can be assigned to a boolean variable, as long as the string or field variable contains a valid boolean value.

**integer**
An integer variable can be assigned an actual integer value or the results of an arithmetic operation. An integer variable must be assigned a valid integer number (for example, 3 or -4). If it is updated with a non-integer value, the decimal portion of the value is truncated (for example, if an integer variable is assigned a value of 4.8, the assigned value is truncated to 4).

**double**
A double variable can be assigned an actual double-precision value or the results of an arithmetic operation when it is created.

**string** A string variable can be assigned combinations of boolean, integer, double, string, or field variables, and actual text strings.

**field** A field position consists of two integers separated by a comma (for example, 2,3). After a field position value is provided by the <varupdate> tag, the text of the field containing the specified field position is assigned to the field variable at run-time. If the value given for the field position does not evaluate to "integer, integer" at runtime, the field's value is an empty string. A run-time error also occurs if a field variable is used before a value is assigned to it with the <varupdate> tag.

### Example

```
<screen>
  <actions>
    <varupdate name="$var_boolean$" value="false"/>
    <varupdate name="$var_int$" value="5"/>
    <varupdate name="$var_double$" value="5"/>
    <varupdate name="$var_string$" value="new_texts"/>
    <varupdate name="$var_field$" value="4,5"/>
  </actions>
</screen>
```

## <playmacro> tag

Runs a macro from within another macro. This process is called chaining. The currently running macro (the parent macro) stops and the macro specified in the

<playmacro> element (the child macro) begins playing. Only macros that are available to run in a particular session can be chained.

The <playmacro> tag must be the last action within the same screen. Any actions after a <playmacro> tag will cause errors to occur. Any actions after a <playmacro> tag will cause errors to occur. The exception is if a <playmacro> tag is contained within an <if>-<else > block (that is, a condition must be satisfied for the macro to play). You can include as many <playmacro> tags in a screen as you like as long as each one is contained within an <if>-<else > block. Each <if>-<else > block can only contain one <playmacro> tag and the <playmacro> tag must be the last action in the block. Control immediately passes to the child macro if it is executed from within an <if>-<else > block; subsequent tags in the parent macro are ignored.

If you wish to chain macros in an application that uses the Host On-Demand beans or HACL APIs, you need to do the following:

- Only managed macros can be chained, so you need to use the MacroManager bean or implement your own MacroIOProvider class.
- Because macros are chained by macro name, you must assign a name to each macro that is to be chained.

**Note:** You cannot use variables as the values for <playmacro> tag attributes.

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

**name** The name of the macro to be played. This can be any valid unicode character.

**startscreen**
The name of the screen at which the macro starts. If a start screen is specified with *DEFAULT* or is not specified, the macro starts at the normal macro entry screen. This attribute is optional.

**transfervars**
Whether the played macro inherits variables from the parent macro. If this value is set to “Transfer”, the variables in the parent macro (if any) are transferred to the child macro and are accessible from the child macro. If it is set to “No Transfer”, the child macro cannot access the parent macro’s current variables. This attribute is optional. The default value is “No Transfer”.

**Notes:**

1. The child macro must set the usevars attribute of its <HAScript> tag to true in order to inherit variables from the parent macro.
2. If the parent and child macro both have variables with the same name, the child macro’s variable is used in the child macro.

### Example

```
<actions>
  <playmacro name="Macro1" startscreen="intro_screen" transfervars="Transfer" />
</actions>
```

## <if> tag

Allows the macro to perform operations based on the truth or falsehood of some condition. If the condition evaluates to true, the operations within the <if> block are performed. If it evaluates to false, they are not. Optionally, an <else> tag can be

used with an <if> block to specify operations to be performed if the <if> statement evaluates to false. See "Using conditional (if-else) statements" on page 154 for more information and examples.

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

**condition**

Specifies the condition (or conditions) to be evaluated. Each condition must resolve to a boolean value. Individual conditions are enclosed in parentheses. You can assign boolean variables or boolean values as the conditions of an <if> element. In addition, the following equality and relational operators are supported:

- = =(Equal to)
- != (Not equal to)
- < (Less than)
- > (Greater than)
- <= (Less than or equal to)
- >= (Greater than or equal to)
- ! (Not)

Conditions are evaluated from left to right. The operators && (logical AND) and || (logical OR) can be used between conditions to perform logical operations on conditional statements. If you are using a code editor to edit the macro, you may need to enter && as &&. See "Using variables" on page 147 for instructions on how to use the reserved characters single quote (') and backslash (\).

### Example

```
<actions>
  <if conditions="($var_int$ == 1) || ($var_bool)">
      ... # Perform macro operations
  </if>
</actions>
```

## <else> tag

An <else> tag can be used with an <if> tag to specify operations that are performed if the <if> conditional statement evaluates to false. This tag can only be used immediately after an <if> tag. See "Using conditional (if-else) statements" on page 154 for more information and examples.

### Example

```
<actions>
  <if conditions="($var_int$ > 10>
      ... # Perform macro operations if $var_int$ is greater than 10
  </if>
  <else>
      ... # Perform other macro operations if $var_int$ is less than
          # or equal to 10
  </else>
</actions>
```

## <runprogram> tag

The <runprogram> tag runs an application from a macro.

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

**exe** Specifies the full path name of the application to be run. The same types of applications that run from the Custom Toolbar Application can be run from a macro's <runprogram> element.

**Note:** See "Using variables" on page 147 for instructions on how to use the reserved characters single quote (') and backslash (\).

**param** Passes a parameter (such as a file name) to the application. The name must be a string enclosed in single quotes (') or a string variable.

**wait** If this parameter is set to true, the macro waits for the application to finish running before resuming play. The default value is false.

**assignexitvalue**
Assigns the exit value of the application to a variable. The value of the wait attribute must be set to "true" in order to use the assignexitvalue attribute.

### Example

```
<runprogram exe="C:\Program Files\Windows NT\Accessories\wordpad.exe"
              param="new_file.doc" wait="true"
              assignexitvalue="$exitstatus$" />
```

## <nextscreens> tag

The **<nextscreens>** tag contains all the valid next screens to be recognized after the current screen's actions have been executed.

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

**timeout**
The allowable time in milliseconds that can elapse between current screen and any next screen before the macro bean will go into the error state. This overrides the timeout attribute for the entire macro. The value must be a number or an expression that evaluates to a number. This attribute is optional. The default is to use the overall macro timeout.

### Example

```
<nextscreens> ... </nextscreens>
```

## <nextscreen> tag

The **<nextscreen>** tag forces a next screen. Multiple **<nextscreen>** tags are allowed. If a screen appears that is in the macro but is not a next screen, the macro will go into an error state. If the next screen refers to a screen tag that doesn't exist, the macro will have a parse error.

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

**name** The name of the <screen> tag that is the valid next screen. This can be any valid Unicode character. This is a required attribute.

### Example

```
<nextscreen name="screen1" />
```

## <recolimit> tag

The **<recolimit>** tag is for advanced use only. It is used to enforce a limited amount of time a screen can be recognized **in a row** before it goes to the screen indicated in the goto attribute. This tag is useful for screen looping where you

know exactly how many times you'll see a given screen in a row. It also is a safeguard against infinite screen recognition.

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

**value** The allowable number of times to recognize a screen. This value must be a number or an expression that evaluates to a number. This is a required attribute.

**Note:** The actions will not be executed the last time the screen is recognized.

**goto** The name of the screen to go to when recognition limit has been reached. This can be any valid Unicode character but the screen must exist in the macro. For Host Publisher, this attribute is optional. If no goto screen is given, the macro terminates.

### Example

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

## Advanced Screen Recognition

The <description> tag defines the recognition criteria for a screen. The optional attribute of other elements within the <description> tag allows you to constrain the descriptors in a limited way. For example, consider the following description:

```
<description>
 <oia status="NOTINHIBITED" optional="false" />
 <string value="aaaaaaaaaaa" optional="false" />
 <string value="bbbbbbbbbbb" optional="true" />
</description>
```

This screen description matches a screen with an OIA status of NOTINHIBITED and a string value of "aaaaaaaaaaa" OR a screen with a string vaue of "bbbbbbbbbbb". The screen recognition logic first tries to see if *all* the optional="false" descriptors are satisfied. Otherwise it attempts to match *any one* optional="true" descriptor.

You cannot use the optional attribute to decribe a screen as follows: OIA status NOTINHIBITED and string "aaaaaaaaaaa" OR *not* oia NOTINHIBITED and string "bbbbbbbbbbb".

Optionally, you can set up more sophisticated matching conditions for screen descriptions by using the uselogic attribute of the <description> tag. This optional attribute allows you to match screens by specifying logical relationships among the screen descriptors such as the following:

```
<description uselogic="(1 and 2) or (!1 and 3)" />
 <oia status="NOTINHIBITED" />
 <string value="aaaaaaaaaaa" />
 <string value="bbbbbbbbbbb" />
</description>
```

The "!" in "(1 and 2) or (!1 and 3)" /> stands for *not*. It is the equivalent of the invertmatch attribute. The numbers represent the descriptors, in the order specified. In this example, the screen description matches if the OIA status is NOTINHIBITED and the string value is "aaaaaaaaaaa" OR the OIA status is *not* NOTINHIBITED and the string value is "bbbbbbbbbbb". The key words AND and OR can be used to represent the logical AND and logical OR operations. These key words are not case sensitive.

Keep the following in mind if you decide to use the uselogic attribute:

- The uselogic attribute is not supported in the Host On-Demand graphical user interface. To use this attribute, you must manually edit a macro file.
- You cannot use variables as the conditions of the uselogic attribute.
- The optional and invertmatch keywords, if specified in a screen descriptor, are ignored.

## Using variables

Variables can be used in macro commands to replace hard-coded values and store the results of operations, just like in any other programming language. They can be created by using the graphical user interface of the Macro Editor or by including the <vars>tag in an existing macro and declaring them with the <create> tag. Variables can also be inherited from other macros; see the description of the <playmacro> tag for details.

To use variables within a macro (including inherited variables), set the usevars attribute of the <HAScript> tag to true. The values of macro tag attributes are then parsed for variable names and arithmetic operators. The value of the usevars attribute is automatically set to false unless a variable was created by using the Host On-Demand Macro Editor. If the value of usevars is false and the macro parser finds a <vars> tag in the macro, the parser displays an error message telling you to set usevars to true.

**Note:** When you create a macro or edit an existing macro in the Macro Editor, the first time that you check ″Use Variables and Arithmetic Expressions in Macro″ in the Macro Editor (setting the usevars attribute of the <HAScript> tag to “true”), Host On-Demand displays a warning that the macro is about to be converted for use with variables and other advanced macro features. (Macros created under Version 6 and earlier of Host On-Demand will still run if they are not converted; the conversion is necessary only to use variables and some other new features introduced in Version 7.)

- If you click OK, usevars is set to true and the macro is converted. All attributes that take string arguments (as opposed to boolean, integer, or keyword arguments) and can be assigned values that are variables or expressions (that is, not the <HAScript> attributes, screen names, or uselogic attributes) are converted. Single quotes are placed around all of the strings that are already assigned to these attributes. For example, if you had an input action with the value “hi”, it becomes “’hi’”. Backslashes are placed in front of existing single quotes in the string (“robin’s” will become “’robin\’s’”), and an extra backslash is placed in front of existing backslashes (“ab\c” will become “’ab\\c’”).
- If you do not click OK, usevars remains set to false and the macro is not converted to the new format. You may choose not to convert if you have already converted your macro, if you would like to convert it by hand, or if you change your mind about using variables. You can then go into the code and set usevars to true yourself.

Variables can be used anywhere within a <screen> element. However, a variable can only be used as a value of an attribute, not as an attribute name. You also cannot use variables to assign values to <HAScript> attributes, <playmacro> attributes, the uselogic attribute of the <description> tag, or macro screen names.

Variable names are specified in a macro as $*varname*$, where *varname* is the name assigned to the variable when it was declared using the <create> tag. The

following example shows how a variable can be created and used to assign a value to an attribute of a macro element (in this case, the <pause> tag):

```
<HAScript usevars="true">
  <vars>
     <create type="integer" name="$pause_length$" value="1000"/>
  </vars>
  <screen>
     ...
     <pause value="$pause_length$">
     ...
  </screen>
</HAScript>
```

Variables that are not defined within a macro can be assigned as attribute values because the variables may be inherited from a parent macro (see the description of the <playmacro> tag for details). However, when you create a variable, you cannot set its value to that of an inherited variable because variables are created and initialized when the macro is parsed, not at run-time.

## Variable types

The following types of variables are supported:

**boolean**
Represents boolean values. A boolean variable can be assigned a value of either “true” or “false”. Note that boolean variables are not case sensitive: values such as “False” or “FALSE” are valid. A string variable or a field variable can be assigned to a boolean variable, as long as the string or field variable contains a valid boolean value. The default value is false.

**integer**
Represents integer numbers. An integer variable must be assigned a valid integer number (for example, 3 or -4). If it is updated with a non-integer value, the decimal portion of the value is truncated (for example, if an integer variable is assigned a value of 4.8, the assigned value is truncated to 4). An integer variable can be assigned an actual integer value or the results of an arithmetic operation. The default value is 0.

**double**
Represents double-precision numbers. A double variable can be assigned an actual double-precision value or the results of an arithmetic operation when it is created. The default value is 0.0.

**string** Represents text strings. A string variable can be assigned combinations of boolean, integer, double, string, or field variables, and actual text strings. String values must be enclosed in single quotes (’). The default value is “”, (an empty string).

**field** Represents text entered into a field on the terminal screen. The position within the field is given as two integers separated by a comma (for example, 2,3). Initially, field variables are empty. A field position must be provided by using the <varupdate> tag, the <prompt> tag, or the <extract> tag (if it is extracting “integer, integer” values from the screen). The text of the field containing the specified field position is assigned to the field variable at run-time. If the value given for the field position does not evaluate to “integer, integer” at runtime, a runtime error occurs unless the field’s value has been set to null (””). If a field variable is used before a value is assigned to it, its value is automatically set to null (””).

## Updating variables

Variable values can be updated in four different ways:

- By using the <varupdate> tag to modify the values of variables after they are created. The entered value must be of the same data type as the variable; otherwise, an error or an unexpected result occurs.
- By using the **assigntovar** attribute of the <prompt> tag to assign user input to a variable. The entered value must be of the same data type as the variable; otherwise, an error or an unexpected result occurs.
- By using the **assigntovar** attribute of the <extract> tag to assign screen text to a variable. This is generally used to store data that is displayed on a terminal emulator screen (for example, the results of a database query) in a variable for later use in the macro. You need to specify the starting and ending rows and columns of the data. If the variable is either an integer or a double, the extracted text must be of a matching data type; otherwise an error or unexpected result occurs.
- By using the **assignexitvalue** of the <runprogram> tag to assign the exit value of an application that has been launched from the macro to a variable.

Updating variables is especially important for field variables, which cannot be given a default value and must be assigned a screen position at run-time.

The following example shows the different ways that variables can be updated.

```
<HAScript usevars="true">
  <vars>
     <create name="$var_bool$" type=boolean value="true"/>
     <create name="$var_int$" type="integer" value="1"/>
     <create name="$final_count$" type="integer"/>
     <create name="$var_double$" type="double" value="1.0"/>
     <create name="$var_string$" type="string" value="some_texts"/>
     <create name="$var_field$" type="field"/>
  </vars>
  <screen>
    <description>
       <varupdate name="$var_bool$" value="false" />
    </description>
  </screen>
  <actions>
    <prompt name="textstring" row="1" col="1" len="72"
        description="Enter a text string"
        clearfield="true" encrypted="false"
        assigntovar=$var_string$/>
    <prompt name="intnumber" row="2" col="1" len="24"
        description="Enter an integer"
        clearfield="true" encrypted="false"
        assigntovar=$var_int$/>
    <varupdate name="$var_field$" value="4,5" />
    <extract name="Get Double value" srow="4" scol="1"
        erow="4" ecol="18" assigntovar=$var_double$ />
    <runprogram exe="C:\myapps\counter.exe"
        wait="true" assignexitvalue="$final_count$"/>
  </actions>
</HAScript>
```

## Arithmetic operations

Arithemetic operations can be performed on numbers, integer variables, double variables, field variable, and string variables. Boolean variables can be used in concatenation operations. The following operations are supported:

- + - Add
  - If the + operator is used on two numbers, they are added.
  - If the + operator is used on two strings or a string and a number, the operands are concatenated.

- - - Subtract
- * - Multiply
- / - Divide
- % - Mod

Parentheses can be used in expressions.

To display these characters on the screen, specify them as literal strings enclosed in single quotes (') - for example, to display the plus sign (+) on the screen, specify the string '+' in the macro. (For instructions on how to use the reserved characters single quote (') and backslash (\), see "Using Special Characters in Macros" on page 157.

Operator precedence is as follows:

1. Parentheses. Operations within parentheses are evaluated first, then the result is used in any subsequent operations.
2. Multiply, divide, mod
3. Add, concatenate, subtract

The following examples show how to use arithmetic operations on variables. Each example gives the syntax of the operation and shows its result. In these examples, the name of the variable is also what the variable evaluates to (for example., $5$ has a value of 5).

```
"(1 + 2) + ', ' + (3 + 5)" = 3, 8
"'Hello ' + $Fred$ + '!'" = Hello Fred!
"$Hi$ $There$" = Error, need a + sign to join strings
"$Hi$+$There$" = HiThere
"$Hi$+'+'+$There$" = Hi+There
"'8.13' + 12" = 8.1312 ('8.13' is a string)
"1 + 2 * 5" = 11
"(1 + 2) * 5" = 15
"10 - (2 / 4)" = 9.5
"(10 - 2) / 4" = 2
"11 / 4" = 2.75
"11 % 4" = 3
"11.0 / 4" = 2.75
"11 / 4.0" = 2.75
"11.0 % 4" = 3.0
"'abc1.08e4' + 3.4e5" = abc1.08e4340000.0
"'5*3'" ="5*3"
"5 + $3$" = 53, where $3$ is a string variable
"5 + $3$" = 8, where $3$ is an integer variable
"'abc\\de'" = abc\de
"'that\'s'" = that's
```

The following example shows some of the ways that arithmetic operations can be used when creating, using, and updating variables.

```
<vars>
   <create name="$var_boolean$" type="boolean" value="false" />
   <create name="$var_int$" type="integer" value="100" />
   <create name="$var2_int$" type="integer" value="$var_int$*5" />
     #OK to use $var_int$ since it is defined already. Result = 500
   <create name="$var_double$" type="double" value="100.9" />
     #Result = 100.9
   <create name="$var2_double$" type="double" value="$var_int$*5" />
     #Result = 500.0
   <create name="$var_string$" type="string">
   <create name="$var1_string$" type="string" value="FirstString" />
</vars>
```

```
<screen>
  <description>
     <varupdate name="$var_string$" value="$var_boolean$$var1_string$++" />
       #Result = "falseFirstString+"
     <varupdate name="$var2_int$" value="$var_int$+400" />
       #Result = 500
  </description>
  <actions>
    <varupdate name="$var2_double$" value="$var2_int$" />
      #Result = 500.0
    <varupdate name="$var_int$" value="$var_double$" />
      #Result = 100 (only gets a whole number part)
    <varupdate name="$var2_int$" value="$var_double$*2" />
      #Result = 100.9 *2 = 201 (not 201.8 or 202)
     <pause value="$var_int$" />
       #OK
     <pause value="$var_int$* 5" />
       #OK
    </actions>
</screen>
```

Be aware of the following:

- If a string or field variable is used as part of an arithmetic expression (not including concatenation), the user must make sure that the string or field variable contains only a numeric value; otherwise an error or an unexpected result can occur at run-time.
- When a double value is assigned to an integer variable, only its whole number value is assigned to the integer. Whatever is after the decimal point is truncated.

## Debugging Variables With MacroActionTrace

If you find that your variables do not seem to contain the right value when you use them, but cannot print the value with an <input> action without throwing off your macro, use MacroActionTrace to display their current values. MacroActionTrace displays variables with their current values as follows:

```
<vars>
<create name="$var1$" type="string" value="'original'" />
</vars>
.
.
.
<actions>
<trace type="SYSOUT" value="'Before update: '+$var1$" />
<varupdate name="$var1$" value="'updated'" />
<trace type="SYSOUT" value="'After update: '+$var1$" />
</actions>
```

This prints the following to the Java console:

```
Before update: {$var1$ = original}
After update: {$var1$ = updated}
```

## Using variables in programmed macros

New methods and classes have been added to the Host Access Beans and Host Access Class Library to allow the use of variables and arithmetic expressions in programmed macros. The following example is a macro that prompts for the user's ID and password, logs the user on to the host, and says "Welcome!":

```
<HAScript name="Logon" description="" timeout="60000" pausetime="300"
          promptall="true" author="" creationdate="" supressclearevents="false"
          usevars="true" >

  <screen name="Screen1" entryscreen="true" exitscreen="false" transient="false">

    <description>
```

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

    <actions>
      <prompt name="'UserID:'" description="" row="20" col="16" len="8"
              default="" clearfield="false" encrypted="false" movecursor="true"
              xlatehostkeys="true" assigntovar="" varupdateonly="false" />
      <input value="'[tab]'" row="0" col="0" movecursor="true"
              xlatehostkeys="true" encrypted="false" />
      <prompt name="'Password:'" description="" row="21" col="16" len="8"
              default="" clearfield="false" encrypted="true" movecursor="true"
              xlatehostkeys="true" assigntovar="" varupdateonly="false" />
      <input value="'[enter]'" row="0" col="0" movecursor="true"
              xlatehostkeys="true" encrypted="false" />
    </actions>
    <nextscreens timeout="0" >
      <nextscreen name="Screen2" />
    </nextscreens>
  </screen>

  <screen name="Screen2" entryscreen="false" exitscreen="true" transient="false">
    <description>
      <oia status="NOTINHIBITED" optional="false" invertmatch="false" />

      <numfields number="7" optional="false" invertmatch="false" />
      <numinputfields number="1" optional="false" invertmatch="false"
    </description>
    <actions>
      <message title="" value="'Welcome!'" />
    </actions>
    <nextscreens timeout="0" >
    </nextscreens>
  </screen>

</HAScript>
```

Assume that you want to use this macro in a Host Access Beans program and you want to store the user ID into a variable and save for later use (for example, in the Welcome message). You could do this directly by modifying the macro, but one reason for doing this programmatically would be to avoid having to maintain many different macros for different situations. You could instead have a base "skeletal" macro and modify it programmatically depending on the situation. The following is an example of how you can do this:

```
// Assume macro is an instantiated Macro with the appropriate listeners set up.
// (See the Javadoc for the Macro bean and the Macro variables demo program,
// MacroVariablesDemo.java, in the Host Access Toolkit samples directory
// for details.)
// Assume macroString is a String containing the previous macro script

macro.setMacro(macroString);
MacroScreens ms = macro.getParsedMacro();
//creates a variable $userid$ with initial value of ""
ms.createVariableString("$userid$", null);
//get the first screen
MacroScreen mscrn = ms.get(0);
//get the actions from the first screen
MacroActions mas = mscrn.getActions();
//get the first prompt action
MacroActionPrompt map = (MacroActionPrompt)mas.get(0);
//assign the prompt response to the variable $userid$
map.setAssignToVar("$userid$");
//get the second screen
MacroScreen mscrn2 = ms.get(1);
//get the actions from the second screen
MacroActions mas2 = mscrn2.getActions();
```

```
//get the message action
MacroActionMessage mam = (MacroActionMessage)mas2.get(0);
//change the message to now be a personalized message using $userid$
mam.setMessage("'Welcome ' + $userid$ + '!'");
//reset the macro with the updated MacroScreens
macro.setParsedMacro(ms);
//play the macro with the changes for variables
macro.play();
```

Suppose you now want to add a second message to the actions for Screen2. In this message, you want to display the time and date, which you extract from the screen. You would add the following lines before `macro.setParsedMacro(ms)`:

```
//create a variable $datetimestamp$ with initial value ""
ms.createVariableString("$datetimestamp$", null);
//create new extract to get date and time from second row of screen
MacroActionExtract mae = new MacroActionExtract(2, 35, 2, 71, "'datetimeextract'");
//assign the date and time string to $datetimestamp$
mae.setAssignToVar("$datetimestamp$");
//add the extract after the first message
mas2.add(mae);
//create a new message to //display the date and //timestamp
mas2.add(mae);
MacroActionMessage mam2 = new MacroActionMessage("'You have logged on at '
          + $datetimestamp$", "'Date Time Stamp'");
//add the message after the extract
mas2.add(mam2);
```

Note that at the point when the attribute containing the variable(s) is associated with the MacroScreens, you must have already created the variable (through one of the createVariable() methods). For example, this code sequence would also be valid:

```
MacroActionExtract mae = new MacroActionExtract(2, 35, 2, 71, "'datetimeextract'");
mae.setAssignToVar("$datetimestamp$");
ms.createVariableString("$datetimestamp$", null);
mas2.add(mae);
MacroActionMessage mam2 = new MacroActionMessage("'You have logged on at
                          ' + $datetimestamp$", "'Date Time Stamp'");
mas2.add(mam2);
```

The above sequence is valid because $datetimestamp$ is created before the MacroActionExtract is added to the MacroActions (which are already associated with the MacroScreens because they were pulled from the MacroScreens originally). If the createVariable() method was called at the end of the sequence above, you would have an invalid sequence because the variable $datetimestamp$ would not have been available at the time that the MacroActionExtract and MacroActionMessage were added to the MacroActions and associated with the MacroScreens.

The default value of the MacroScreens method isUseVars() is false. However, if you call one of the createVariable() methods on your MacroScreens, isUseVars() will return true automatically. If you don't create any variables, but want to have your attributes scanned for variables and arithmetic anyway (for example, you may be writing a chained child macro that has no variables of its own but is anticipating some from the parent), you must call setUseVars(true) on your MacroScreens.

Attributes that can now take variables or expressions as arguments have setAttribute(String) and either getAttributeRaw() or isAttributeRaw() methods available. If you wanted to use an expression now to represent the row attribute for a MacroActionInput, you could call setRow("$rowvar$ + 1"). Subsequently calling getRow() would return the evaluated value of this expression (an integer),

whereas calling getRowRaw() would return “$rowvar$ + 1.” Note that if you do the following you will get a NumberFormatException:

```
MacroActionInput mai = new MacroActionInput();
mai.setRow("$rowvar$ + 1");
int row = mai.getRow();
```

This is because mai has not yet been associated with any MacroScreens with isUseVars() returning true. Therefore, “$rowvar$ + 1.” is being treated as a string rather than a variable plus one. Note also that if you had call the setAttribute() methods to set up variables and expressions *after* the object containing these attributes have been associated with the MacroScreens, you will likely experience a savings in processing time as the attributes would otherwise need to be reparsed for variables/expressions at the point when they are added to the MacroScreens.

The VariableException class is available for catching exceptions such as illegal expressions (for example, ″45 *″) or illegal arithmetic operands (for example, ″’3a’ * 2″).

A sample program that uses programmed macros, MacroVariablesDemo.java, can be found in the Host Access Toolkit samples directory. See the readme.txt file for instructions on how to use this sample application.

## Using conditional (if-else) statements

You can use <if> and <else> tags to create conditional statements in macros. A conditional statement performs operations based on whether a certain condition (or set of conditions) evaluates to true or false. If the condition evaluates to true, the operations within the <if> tag block are performed. If it evaluates to false, they are not. Optionally, an <else> tag block can be used with an <if> block to specify operations to be performed if the <if> statement evaluates to false.

Conditional <if> - <else> statements are used in <actions> tag blocks and provide selection structures to control program flow. When used in conjunction with variables, they are a powerful tool for creating sophisticated macros.

Every <if> statement specifies a condition (or set of conditions) to be evaluated at macro run-time. The entire set of conditions must resolve to a boolean value. Individual conditions are enclosed in parentheses (). You can assign boolean variables or boolean values as the conditions of an <if> element. In addition, the following equality and relational operators are supported:

- = =(Equal to)
- != (Not equal to)
- < (Less than)
- > (Greater than)
- <= (Less than or equal to)
- >= (Greater than or equal to)

Conditions are evaluated from left to right as follows:

- A number compared to number is a number compare.
- A string compared to string is a string compare.
- A string compared to a number is a string compare.
- A field variable is treated first as a number; if it is not numeric, it is treated as a string.

For example, the following conditional evaluates to false because a string is being compared to a number:

```
'4.2e2' == 420
```

The following conditional evaluates to true because a number is being compared to a number:

```
4.2e2 == 420
```

The operators && (logical AND) and || (logical OR) can be used between conditions to evaluate their logical relationships. If you are entering && through a code editor, you may need to enter it as &&.

Parentheses are used to nest conditions. If you want to use an arithmetic expression with nested expressions (that is, an expression containing parentheses) as part of your condition, you must first assign that expression to a variable and use the variable instead THe following example is invalid:

```
(5 + 2) * 3 == 21
```

Instead, state the condition as follows:

```
$expression$ == 21
```

where $expression$ is a variable that was updated with the value (5 + 2) * 3.

The following example shows an <if> tag with a single condition:

```
<vars>
   <create name="$var_string$" type="string" value="no"/>
</vars>
<screen>
  <description>
      ...  #Screen operations
  </description>
</screen>
<actions>
  <prompt name="filesend" row="1" col="1" len="10"
          description="Send a file? (yes/no)"
          clearfield="true" encrypted="false"
          assignto=$var_string$/>
  <if condition="($var_string$ == yes)">
      <input value="[clear]"/>
      <filexfer direction="send.txt" pcfile="myfile.txt"
        pcfile="myfile text a0" />
  </if>
</actions>
```

The following example shows an <if> - <else> statement with multiple conditions:

```
<vars>
    <create name="$condition1$" type="string"/>
    <create name="$condition2$" type="boolean" value="false"/>
    <create name="$condition3$" type="integer"/>
</vars>
<screen>
   <description>
        ... # Screen elements
   </description>
   <actions promptall="true">
        <extract name="Get condition 1" srow="2" scol="1" erow="2"
                 ecol="80" assigntovar="$condition1$"/>
        <extract name="Get condition 2" srow="3" scol="1" erow="3"
                 ecol="80" assigntovar="$condition2$"/>
        <extract name="Get condition 3" srow="4" scol="1" erow="4"
```

```
                ecol="80" assigntovar="$condition3$"/>

 <if conditions="(($condition1$ !="")&&($condition2])||($condition3$ < 100))">
             ... # Perform one set of macro actions if $condition1$ is
                 # not an empty string and $condition2$ evaluates to
                 # "true", or if the value of $condition3$ is less
                 # than 100
        </if>
        <else>
             ... # Perform a different set of macro actions if the
                 # <if> element conditions are not met.
        </else>
   </actions>
</screen>
```

## Converting Numbers to and from the Local National Language Format

Different NLS locales represent numbers in different ways. For example, a decimal number such as 1234.56 can be represented as 1,234.56, 1234.56, or 1234,56 depending on where you are located. Similarly, depending on the locale, negative numbers can be indicated with a minus sign either before or after the number (for example, -78 or 78-).

To enable macros to make use of numbers in local formats, Host On-Demand supplies two conversion methods:

- $FormatStringToNumber(*value*)$, which converts a value in a local format to a double-precision number.
- $FormatNumberToString(*value*)$, which converts a number to its local format value.

The local format is determined by checking the NLS locale of the current Host On-Demand session. The <HAScript> **usevars** attribute must be set to “true” to use these conversion methods. Conversion methods can be nested.

These conversion methods can be used as the values of macro attributes in the same way as variables. As their parameters, they can take numeric values, strings, or variable expressions that evaluate to a numeric value or string. Parentheses are reserved. To specify an argument string containing one of these characters (for example, an arithmetic expression), assign the argument string to a variable first, then use the variable as the argument.

In the following example, the value 3.24 is converted to its local equivalent (for example, 3,24) when it is sent to the screen:

```
<input value="$FormatNumberToString(3.24)$" row="1" col="1"
          movecursor="true" xlatehostkeys="false" />
```

In the following example, the FormatStringToNumber method is used as a parameter of the FormatNumberToString method, converting the number $num$ from its local NLS format before performing the arithmetic operation:

```
$FormatNumberToString(1000 * $FormatStringToNumber($num$)$)$
```

In the following example, a user is extracting a negative integer value such as 3- from the screen. To find out whether the extracted value is really a negative number, it must first be converted to the standard representation of -3 by using the FormatStringToNumber method. The variable $value$ is a string variable that holds whatever is extracted from the screen.

```
<extract name="Extract" planetype="TEXT_PLANE" srow="1" scol="1"
         erow="-1" ecol="-1" unwrap="false" assigntovar="$value$" />

<if condition="($FormatStringToNumber($value$)$< 0)">
              #converting 3- to -3 and comparing its value

         ...  #Perform macro actions
</if>
```

**Using Special Characters in Macros:** Certain characters in Host On-Demand macros have special functions depending on whether variables are in use. To use these characters in a macro, you must enter them in a specific format in the macro code. This section describes the different ways in which, depending on the situation, various special (or reserved) characters are specified in macros.

**If variables are in use**

If variables are in use in the macro (that is, if the <HAScript> tag's usevars attribute is set to ″true″), special characters are used in a macro as follows:

- \ - Used to identify special characters. If you want to write a ″\″ character to the screen, you must enter \\ in the macro. For example, 'C:\\myfile.txt' writes ″C:\myfile.txt″ to the screen.
- ' - Used to identify literal strings (such as 'some text'). If you want to actually write a ″'″ character to the screen, you must enter \' in the macro. For example, 'it\'s' writes ″it's″ to the screen.
- The characters +, -, *, /, %, >, <, =, & and | are evaluated as operators unless enclosed in single quotes (') as part of a literal string.
- The characters $x$ identify a variable or method (where x is the variable's name, such as $MyVariable$) unless enclosed in single quotes (') as part of a literal string.

**If variables are not in use**

If variables are not in use (that is, if the <HAScript> tag's usevars attribute is not set to ″true″), special characters are treated as follows:

- The single quote and backslash characters are not treated specially. For example, C:\myfile.txt writes ″C:\myfile.txt″ to the screen and it's writes ″it's″ to the screen.
- The characters +, -, *, /, %, $, >, <, =, & and | are evaluated as literal characters, not operators.

# Appendix D. Notices

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

**The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:**
INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
TL3B/062
3039 Cornwallis Road
RTP, NC 27709-2195
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

## Programming interface information

This Developer's Guide contains information on intended programming interfaces that allow the customer to write programs to obtain the services of Host Access Transformation Server.

## Trademarks

The following terms are trademarks of the IBM Corporation in the United States or other countries or both:

- IBM
- WebSphere

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

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

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

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

# Glossary

This glossary lists some of the terms used in this book along with their meanings.

**action** A step to be executed when an event occurs, such as a host screen matching the screen recognition criteria specified for a screen customization. A list of actions is part of the definition of each event, including the screen customization resource.

**application**
See **HATS application**.

**application keypad**
A set of buttons or links representing application-level functions. (Contrast with **host keypad**.)

**assemble**
To collect the resources of a HATS project, along with the necessary executable code, into an application .ear file in preparation for transferring the application to the server.

**business logic**
Any Java program invoked as an action in an event, such as a screen customization. Business logic is specific to the application and is not provided as part of HATS.

**component**
A visual element of a host screen, such as a command line or menu. HATS applications transform host components into widgets.

**deploy**
To make a HATS application ready for use on the server, using functions in WebSphere Application Server, after transfer has taken place. Note that WebSphere documentation sometimes uses the term *install* as a synonym for this process. (See also **transfer**.)

**developer**
The person who uses HATS Studio to develop applications; also application developer or Web developer. (Contrast with **user**.)

**.ear file**
Enterprise Archive. The J2EE-format file containing the project resources and executable code for a HATS application.

**Eclipse**
The open-source implementation upon which WebSphere Studio is built and which is available for download from http://www.eclipse.org.

**editor** An application that enables a user to modify existing data. In HATS Studio, editors are used to customize resources that have been created by wizards.

**event** An occurrence of interest to a program, for which a specific response in the form of one or more actions is defined. The matching of a host screen by a screen customization's screen recognition criteria is an event, as are a user connecting to or disconnecting from a HATS application, and the failure of a host screen to match any screen customizations.

**Extensible Markup Language (XML)**
A standard metalanguage for defining markup languages that was derived from and is a subset of SGML.

**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, or business logic.

**HATS** See **Host Access Transformation Server**.

**HATS application**
An application that presents a Web-enabled version of a host application to users. A HATS application is created in HATS Studio from a HATS project and deployed to WebSphere Application Server.

**HATS project**
A collection of resources (also sometimes called ″artifacts″) created and customized in HATS Studio, which can be assembled into a HATS application.

**HATS Studio**
The component of HATS that runs on WebSphere Studio and enables you to work with HATS projects to create HATS applications.

**Host Access Transformation Server (HATS)**
An IBM software product that enables you to present host applications as Web-based applications.

**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**.)

**HTML**
Hypertext Markup Language.

**J2EE** Java 2 Platform, Enterprise Edition. An environment for developing and deploying enterprise applications, defined by Sun Microsystems Inc. The J2EE platform consists of a set of services, application programming interfaces (APIs), and protocols that provide the functionality for developing multitiered, Web-based applications.

**JavaServer Pages (JSP)**
A server-side scripting technology that enables Java code to be dynamically embedded within Web pages (HTML files) and executed when the page is served, returning dynamic content to a client.

**JSP** See **JavaServer Pages**.

**macro** An XML script that defines a set of screens. Each screen includes a description of the screen, the actions to perform for that screen, and the screen or screens that can be presented after the actions are performed. A macro can be specified as one of the actions to be taken when a host screen matches the screen recognition criteria of a screen customization.

**perspective**
In the WebSphere Studio 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.

**project**
See **HATS project**.

**resource**
Any of several data structures included in a HATS project. HATS resources include templates, screen customizations, transformations, screen captures, and macros. Other WebSphere Studio plugins sometimes call these ″artifacts.″

**Run On Server**
A function of WebSphere Studio, which enables the developer to test or preview a project using the embedded WebSphere Application Server. Sometimes referred to as ″WAS Test Environment.″

**screen capture**
An XML representation of a host screen, used to create or customize a screen customization or transformation.

**screen customization**
A HATS resource with two parts: a set of screen recognition criteria used to match host screens, and a list of actions to be taken when a host screen matches the screen recognition criteria.

**screen recognition criteria**
A set of criteria used to determine whether a host screen matches a screen customization and should have that screen customization's actions applied.

**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 markup-language files 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**.

**template**
A HATS resource that describes the relatively static portion of the Web pages presented by the HATS application, including a banner and navigation area.

**transfer**
To copy an application .ear file to the server, usually by FTP. (See also **deploy**.)

**transformation**
A HATS resource that specifies how to convert components of a host screen into widgets on a Web page.

**user** The end user of an application that runs on the server. (Contrast with **developer**.)

**WebSphere**
A family of IBM software products that provide a development and deployment environment for basic Web publishing and for transaction-intensive, enterprise-scale e-business applications.

**WebSphere Application Server**
An IBM software product that provides the core software needed to deploy, integrate and manage e-business applications. HATS applications, when assembled and transferred to a server, run as WebSphere Application Server applications.

**WebSphere Studio**
Any of several IBM software products that provide an integrated development environment based on the Eclipse open-source platform.

**widget**
A visual element of a Web page, such as a button, entry field, or drop-down list. HATS applications transform host components into widgets.

**wizard**
An interface that enables you to complete a task in defined steps. HATS uses wizards to create projects and their resources.

**workbench**
Synonym for WebSphere Studio.

**XML** See **Extensible Markup Language**.

# Index

## 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® WebSphere® Host Access Transformation Server**
**Developer's Guide**
**Version 4**

**Publication No. SC31-6324-00**

**Overall, how satisfied are you with the information in this book?**

| | Very Satisfied | Satisfied | Neutral | Dissatisfied | Very Dissatisfied |
|---|---|---|---|---|---|
| Overall satisfaction | ☐ | ☐ | ☐ | ☐ | ☐ |

**How satisfied are you that the information in this book is:**

| | Very Satisfied | Satisfied | Neutral | Dissatisfied | Very Dissatisfied |
|---|---|---|---|---|---|
| Accurate | ☐ | ☐ | ☐ | ☐ | ☐ |
| Complete | ☐ | ☐ | ☐ | ☐ | ☐ |
| Easy to find | ☐ | ☐ | ☐ | ☐ | ☐ |
| Easy to understand | ☐ | ☐ | ☐ | ☐ | ☐ |
| Well organized | ☐ | ☐ | ☐ | ☐ | ☐ |
| Applicable to your tasks | ☐ | ☐ | ☐ | ☐ | ☐ |

**Please tell us how we can improve this book:**

Thank you for your responses. May we contact you? ☐ Yes ☐ No

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you.

Name _____ Address _____

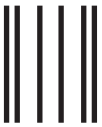
Company or Organization _____

Phone No. _____

**Readers' Comments — We'd Like to Hear from You**
SC31-6324-00

Cut or Fold
Along Line

Fold and Tape | **Please do not staple** | Fold and Tape



NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

# BUSINESS REPLY MAIL

FIRST-CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

IBM Corporation
Software Reengineering
Department G7IA/ Bldg 503
Research Triangle Park, NC 27709-9990

Fold and Tape | **Please do not staple** | Fold and Tape



Cut or Fold
Along Line

Printed in U.S.A.

SC31-6324-00