webMethods JIS:

XHTML Client User’s Guide

Version 9.0

November 2009
(originally released January 2005)

Document ID: JIS-XHTMLCLIENT-UG-90-20121115

Table of Contents

List of Figures

List of Tables

List of Examples

About This Guide

The XHTML client is one of several available client types. This manual specifically covers topics related to the XHTML client. This manual is accompanied by *webMethods JIS: Basic User's Guide*, which covers all topics related to the processing of host application screens and their development.

This manual is structured as follows:

- **Chapter 1 -** "**webMethods JIS: XHTML Client**". An introduction to the product and guidelines for using this manual.
- **Chapter 2 -** "**The Development Environment**". An introduction to the development environment as well as instructions for setting up the development environment on the Development Machine.
- **Chapter 3 -** "**Creating the JIS Application**". Instructions for creating the webMethods JIS Application in ACE
- **Chapter 4 -** "**Deploying the JIS Runtime Application**". Instructions for deploying the webMethods JIS runtime on the Server Machine.
- **Chapter 5 -** "**Optimizing the JIS Server**". Instructions for customizing and optimizing the JIS Server.
- **Chapter 6 -** "**Language Localization**". Information about implementing the localization feature in your application.
- **Chapter 7 -** "**XHTML Runtime Architecture**". An examination of the XHTML runtime components and architecture.
- **Chapter 8 -** "**Enhancing Your Application Using HTML Extensions**". Instructions for creating user HTML extension files and incorporating them into the webMethods JIS Application.
- **Chapter 9 -** "**Enhancing Your Application Using Java Extensions**". A detailed examination of a set of examples for incorporating Java extensions into your application.
- **Chapter 10 -** "**Conducting XML-based Transactions from the Client**". Introduction to XML Transactions and instructions for creating and using them.
- **Chapter 11 -** "**The Server Configuration File**". An introduction to the Server Configuration File and instructions for modifying and configuring it, as well as creating a new Server Configuration File from scratch.
- **Chapter 12 -** "**Application Server Deployment**". An explanation of how to run webMethods JIS applications on a J2EE application server.

Documentation Set

webMethods JIS is supplied with the manuals shown below. The documentation is delivered in Adobe Acrobat Reader Portable Document Format (PDF). No hardcopy documentation is provided, but you can print the PDF files on your local printer.

Table 1. webMethods JIS documentation set

This book...	Contains...
webMethods JIS: Getting Started with the Automated Conversion Environment	Startup information and an introduction to the Automated Conversion Environment (ACE).
webMethods JIS: Basic User's Guide	Full explanations of the ACE Views and how to use them
webMethods JIS: Advanced Topics	Explanations of advanced features that give your application extra functionality.
webMethods JIS: KnowledgeBase User's Guide	In-depth information about the way the ACE KnowledgeBase is designed and how to work with it.
webMethods JIS: Java Client User's Guide	Information for migrating your host application to Java.
webMethods JIS: XHTML Client User's Guide	Information for migrating your host application to an XHTML web application.

Document Conventions

The following conventions are used throughout this manual.

Table 2. Documentation conventions

Convention	Description
Click	Position the mouse pointer on the control and quickly press and release the left mouse button **once**. (Unless the right mouse button is explicitly specified, you should click the left mouse button.)
Double-click	Position the mouse pointer on the control and quickly press and release the left mouse button **twice**. (Unless the right mouse button is explicitly specified, you should double-click the left mouse button.)
UPPERCASE	Uppercase letters are used for the names of files. For example, a panel file with the name Menu, will be expressed as MENU.PNL**.**
italics	Names of applications, programs, menus, dialog boxes, and libraries.
Bold	Menu options, and items, dialog boxes and items to be selected from a dialog box. The names of pull-down menus.
Bold Italics	Pattern definitions, representation definitions, message definitions, method names, layout names, section names, selection definitions, function definitions.
BOLD + UPPERCASE	Keyboard shortcuts: Press the **SHIFT** key. Press **CTRL + Z**.

Viewing the Documentation Online

You can also access the latest version of the documentation for Software AG products at http://documentation.softwareag.com/. As new versions become available, the documentation on this web site will be updated and the previous versions will be migrated to the Software AG documentation web site at http://servline24.softwareag.com/public/. If you have a maintenance contract, you can view all versions of documentation on this web site. You will find instructions for registering and obtaining a userid and password on the documentation web site.

Chapter 1. webMethods JIS: XHTML Client

webMethods JIS is an automated development architecture that generates XHTML clients for enterprise applications. It uses webMethods JIS's Automated Conversion Environment to convert character-based host screens into feature-rich graphical XHTML clients. Consequently, mainframe and iSeries applications can be delivered over the internet using an XHTML-capable client interface.

This chapter discusses the various aspects of the XHTML Client and provides guidelines for using this manual.

The following topics are discussed:

- "JIS Terminology" on page 23
- "Introducing webMethods JIS" on page 25
- "Runtime Architecture" on page 25
- "Runtime Dataflow" on page 28
- "Implementing the XHTML Client" on page 29

JIS Terminology

Working with webMethods JIS, you encounter the following terms:

Table 3. JIS terminology (Sheet 1 of 2)

Term	Description
Host Computer, Host Machine	Runs the Host, a 5250 or 3270-style application. The Host can be either an iSeries or a Mainframe computer.
Host Application	The actual program that operates on the Host computer. A Host computer may have more than one Host application on it.
Client	The XHTML web-based interface through which an end-user connects to the host in order to gain access to Host applications.

Table 3. JIS terminology (Sheet 2 of 2)

Term	Description
Server Machine	The Server computer hosts the webMethods JIS deployment environment.
JIS Server	The JIS Server opens a session on the Host from one side and interacts with the end-user on the other. The JIS Server program must be installed on the Server Machine.
Host session, Session	A live connection to the host. On a Host session, a user may navigate through the different host screens.
Development Machine	The computer that hosts the webMethods JIS Automated Conversion Environment (ACE).
ACE	Automated Conversion Environment - is used to analyze the host interface and create the webMethods JIS development environment.
JIS Application	A collection of all the converted host screens, methods and other attached information developed in ACE.
JIS Runtime, Runtime	The executable form of a completed webMethods JIS application. The runtime is installed on the Server Machine.
ACE Method, Method	A short program comprised of actions and commands. The webMethods JIS Application developer may create new methods or change existing ones. Methods enhance an Application's functionality.
XML Transaction	A concise pre-structured exchange of information between Host and Client. An XML Transaction is comprised of several ACE Methods. It is configured in the Server Configuration File.

Introducing webMethods JIS

webMethods JIS extends the functionality of your legacy application to generate an XHTML Client, so that using a web browser with HTTP connectivity, end-users may connect to your legacy system and browse through it. In this way your end-users receive an attractive and enhanced GUI representation of your legacy system's screens. Through these screens, the client may connect to your legacy system just as a standard "green screen" dumb terminal would.

Problems Solved by the XHTML Client

webMethods JIS specializes in integrating MainFrame and iSeries business logic with dynamic web pages. If your company relies on legacy applications and you wish to extend your connectivity to the internet, then webMethods JIS is your solution.

The webMethods JIS solution:

- Enables organizations to respond more quickly to business opportunities represented by new interface alternatives.
- Eliminates the extraneous expense incurred by IT departments to retool and react to the ever changing interface standards.
- Significantly improves employee productivity by providing a common look and feel across new and existing applications - including seamless desktop integration with today's Customer Relationship Management (CRM) applications.

Runtime Architecture

The following sketch outlines the XHTML Client runtime architecture:

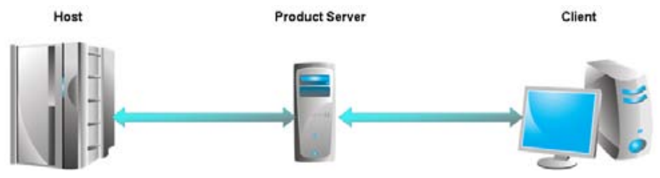


Figure 1. XHTML client runtime architecture

The XHTML Client runtime architecture includes three components:

- The Host
- The JIS Server
- The Client

The Host

The host application resides on a Legacy system. The host application interacts with the JIS Server in the host application's language.

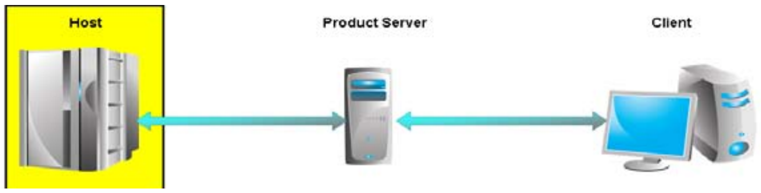


Figure 2. The host

During runtime, the host keeps working just like it did before integration with webMethods JIS, thinking that it is communicating with just another dumb terminal.

The JIS Server

The JIS Server houses the JIS Runtime Application. The JIS Server interacts with the host application on one side and with the end-user on the other.

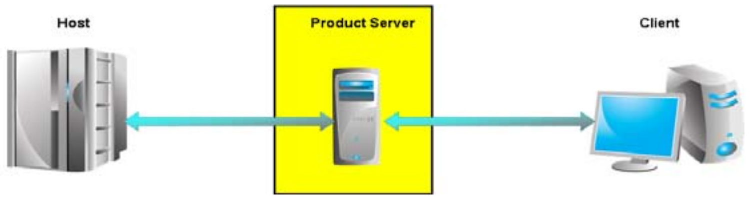


Figure 3. The JIS Server

Information transferred between the JIS Server and the end-user is transferred as XHTML via HTTP. Information transferred between the host application and the JIS Server is transferred using the host system's communication protocol.

The JIS Server resides on the Server Machine. In some forms of deployment the JIS Server may reside together with the host application on the host machine.

Function

During runtime, the JIS Server:

- Communicates with the Host Application by emulating a "green screen" dumb terminal. The JIS Server receives host screens from the host.
- Identifies each received host screen and matches it with a corresponding GUI screen. Such GUI screens are also known as Subapplications.

- Composes each Subapplication together with dynamic information received from the host into an XHTML page known as a Subapplication XHTML.
- Sends Subapplication XHTMLs to the end-user.
- Receives form data from the end-user. This happens each time the end-user submits the form, contained in the last received Subapplication XHTML, back to the JIS Server.
- Updates the host application with new information. This is done by filling in fields and pressing accelerators on the current host screen.

Components

The JIS Server is comprised of files generated from the resulting efforts of the JIS Application Development. Once the JIS runtime Application is fully developed, it is packaged into an executable. This executable is then extracted onto the Server Machine.

The Client

The Client is the XHTML web-based interface, through which the end-user communicates with the host application. The Client interacts with the JIS Server HTTP using a web browser.

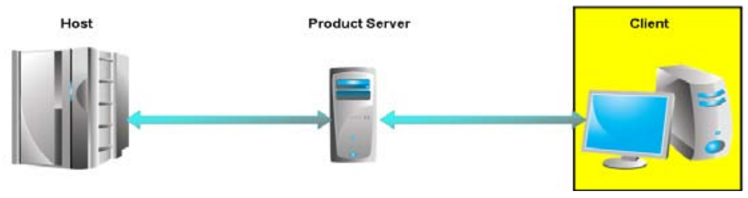


Figure 4. The XHTML client

The Client may work on any computer with an internet connection.

Function

During runtime, the Client:

- Initiates a connection with the host, by sending an HTTP request to the JIS Server through a web server.
- Receives a GUI representation of the host screens in the form of a well-formed HTML.
- Sends HTML form data back to the JIS Server.

Components

The only component the Client requires is a web browser.

Runtime Dataflow

What happens in runtime is described in Figure 5.

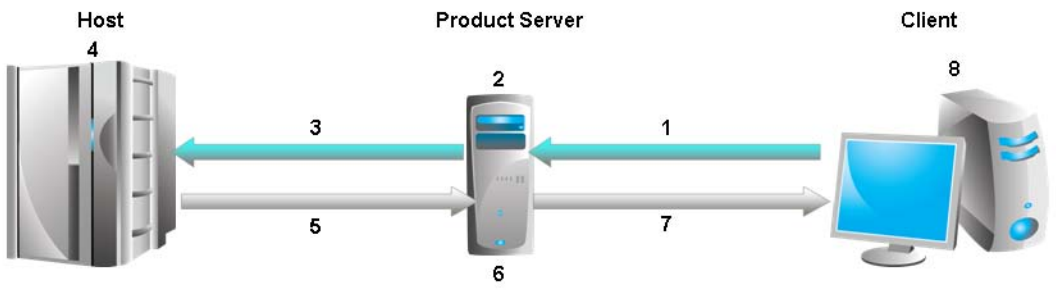


Figure 5. Host-server-client information flow

This is the sequence of events:

1 **The Client sends an HTTP request to the JIS Server.**

After the initial connection, the browser displays the JIS application GUI. The client fills in fields and submits forms via HTTP to the JIS Server.

2 **The JIS Server processes the information submitted from the Client.**

The JIS Server processes the information received from the client and translates it into a language the host can understand.

3 **The JIS Server communicates with the host.**

The JIS Server emulates a standard "green screen" dumb terminal, filling the appropriate fields and pressing the appropriate accelerator keys on the host.

4 **The host updates itself and generates a new screen.**

The host sends the new screen back to the JIS Server.

5 **The JIS Server generates a GUI.**

The JIS Server recognizes the new host screen, matches it with a corresponding GUI equivalent, known as a Subapplication, and updates the Subapplication's fields with dynamic information from the host.

6 **The JIS Server sends a well-formed HTML page to the Client.**

The JIS Server composes the Subapplication into an XHTML page known as a Subapplication XHTML. The JIS Server sends this XHTML page to the Client via HTTP.

7 **The Client displays a GUI representation of the current host screen.**

For more detailed information, see Chapter 7 - "XHTML Runtime Architecture" on page 233.

Implementing the XHTML Client

The workflow for implementing the XHTML Client consists of two main phases:

- The development phase.
- The deployment phase.

Each of these phases requires different prior knowledge and skills.

The Development Phase

In the development phase, the JIS Application is created.

The following actions are performed in this phase:

- Preparing the host screens for conversion.
- Installing ACE on the development machine.
- Creating the JIS Application in ACE.
- Testing the JIS Application.
- Creating an executable.

The Deployment Phase

In the deployment phase, the system, on which the JIS Application will operate, is assembled, fine-tuned and launched.

The following actions are performed in this phase:

- Installing the executable on the server machine.
- Optimizing the JIS Server.
- Enhancing the Application's look and feel.
- Performance testing.
- Launching the finalized product.

Chapter 2. The Development Environment

The JIS Application is developed on a development machine. Most of the development work is conducted using ACE, webMethods JIS's Automated Conversion Environment, which is installed on the development machine.

This chapter provides an overall view of the development environment and the development workflow, listing the various building blocks of the development environment and what is needed in order to set it up the development environment.

The following topics are discussed:

- "Schematic Diagram of Development Architecture" on page 31
- "Software Used for test Development" on page 32
- "Workflow for Developing the JIS Application" on page 33
- "Setting up the Development Environment" on page 34

Schematic Diagram of Development Architecture

The following sketch outlines the development architecture:

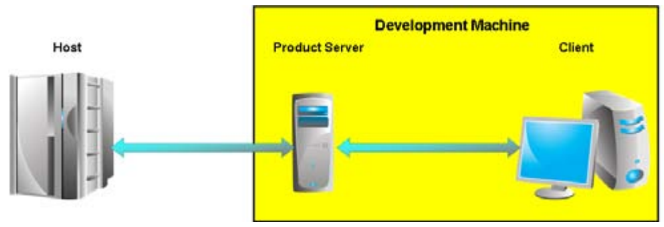


Figure 6. The development architecture

The development architecture consists of two components:

- The host
- The development machine

The Host

While creating the JIS Application, you need to access the host application for the purpose of extracting the host screens for conversion. Once the JIS Application is created, you will need to test it by simulating a live connection with the host.

The Development Machine

The JIS Application is created, tested and packaged for production on the development machine. First, ACE is installed on the development machine. In ACE, you create the JIS Application and create a runtime. Then, you must test the JIS Application created in ACE. To do this, the development machine simulates a live runtime connection with the host, in which both components, JIS Server and Client, are housed in the development machine.

Software Used for test Development

The development environment is comprised of various software components. To set up the development environment, you have to install each one of these.

The following software is used for development:

- ACE
- Java compiler
- WISE InstallBuilder (optional)
- Web server (optional)
- Web browser

ACE

webMethods JIS's Automated Conversion Environment (ACE) is used for converting the host screens into GUI equivalents known as Subapplications and composing these into a JIS Application. You then use ACE to create the distributable, which is later deployed on the server machine.

Java Compiler

To compile your JIS Application you need a Java compiler. This process is known as Runtime Generation. It is an automated process that is performed in ACE using the Java compiler supplied with the ACE installation package.

WISE InstallBuilder

To create the distributable, ACE enables you to create a WISE installation. This is an automated process performed in ACE, but it requires the prior installation of WISE Installation Studio 7.0 or higher on your development machine.

Note: You may choose a different installation packaging tool with which to create the distributable.

Web Server

The JIS Server uses a web server to handle HTTP communication with the client. The Jetty HTTP Server is an integral part of the JIS Server. By default, a JIS Application's GIF files are handled by the Jetty HTTP Server. This is done using the ResourceBase setting in the `[HTTP]` section of the JIS Server INI file. For more information, see "JIS Server INI File Settings" on page 105.

Web Browser

To test the JIS runtime, you will need a web browser that supports HTML 4.

webMethods JIS supports the following web browsers:

- Internet Explorer 7.0 and higher
- Firefox 3.0 and higher

Workflow for Developing the JIS Application

Following is the entire workflow for developing the JIS Application.

To develop the JIS Application:

1. Install ACE on the Development Machine. See "Installing ACE on the Development Machine" on page 34.
2. Install additional software. See "Additional Installations" on page 35.
3. Create the JIS Application in ACE. See "Working in ACE" on page 37.
4. Generate a runtime. See "Generating a Runtime" on page 62.
5. Test the generated runtime. See "Testing the Generated Runtime" on page 67.
6. Create a distributable. See "Creating the Runtime Installation" on page 78.

For information on enhancing your application using HTML extensions, see Chapter 8 - "Enhancing Your Application Using HTML Extensions" on page 243.

For information on enhancing your application using Java extensions, see Chapter 9 - "Enhancing Your Application Using Java Extensions" on page 279.

Setting up the Development Environment

The webMethods JIS Application developer's first task is to prepare the development environment. The development machine must be set up so that you can use it for converting the host screens into GUIs to create the JIS Application, and for simulating a full deployment environment to test the JIS Application.

To set up the development environment:

1. Install ACE on the development machine.
2. Perform additional installations.

Installing ACE on the Development Machine

Before installing ACE make sure you have:

- The webMethods JIS installation CD.
- Your installation code.

Note: ACE requires the Borland Database Engine. This is supplied with the webMethods JIS installation package.

To install ACE on the PC:

1. Turn on the PC. Place the CD in the drive, and run **setup** from the root directory.
2. Follow the directions in the **Setup** wizard. You will be asked to supply the installation code during the course of the installation. Make a note of the installation drive and installation directory name.

Installing Multiple Versions

You can install more than one version of ACE. If you have a previous version of ACE installed on your machine, the **Setup** wizard gives you the choice of either upgrading your existing version or making a separate installation for the new version.

If you choose to upgrade, you will not be able to reinstall the older version unless you first uninstall the upgraded version of ACE.

Additional Installations

After successfully installing ACE on the development machine, you must install an application installation builder, such as Wise InstallBuilder.

In order to set WISE registry entries correctly, open Wise as a stand-alone application. You need to do this only once.

Chapter 3. Creating the JIS Application

The JIS Application is created in webMethods JIS's Automated Conversion Environment (ACE), which comes accompanied by the ACE documentation. These are books, which guide you through every step of the way as you create the JIS Application.

This chapter provides guidelines for creating the JIS Application while using the ACE documentation.

The following topics are discussed:

- "Working in ACE" on page 37
- "Converting the Host Application" on page 39
- "Considerations When Working in ACE" on page 49
- "Wait for Screen" on page 54
- "Generating a Runtime" on page 62
- "Testing the Generated Runtime" on page 67
- "Packing and Unpacking Applications" on page 71
- "Section 508 Guidelines" on page 71

Working in ACE

ACE takes the screens you have chosen in your host application and generates JIS screen-equivalents and fields in an off-line conversion process. These are displayed in ACE in the form of a GUI window, in which any host screen component is displayed as a GUI element. These GUI elements are referred to as "controls". During compilation (Runtime Generation) the converted host screens are compiled into Java classes.

The JIS application's screens and fields are **automatically** generated from each host screen. The automation is based on the proprietary KnowledgeBase which identifies the information present on a host screen.

To work in ACE you must be familiar with the following terms:

- KnowledgeBase
- Screen Image
- Subapplication
- Application
- Library

KnowledgeBase

The KnowledgeBase is a set of rules for reading and understanding host screens. The rules are based on the fact that host applications present information in standard ways.

The KnowledgeBase is both flexible enough to account for wide variation within the standard ways that host screens present information and robust enough to correctly discern between similar character combinations and generate the appropriate field.

Screen Image

A screen image is a "picture" of a host screen. ACE cannot work directly on the host, so for each host screen you must create a screen image file on your development platform. A screen image can be a screen capture—a simple snapshot of the host screen, or the screen image can be an ACE readable form of a host Screen Definition File: Mainframe BMS/MFS or iSeries DDS.

Screen images are created step-by-step in the **Create Screen Images** wizard. The wizard includes complete explanations of how to configure the screen images. You can create more than one screen image from a single host screen or SDF.

Subapplication

A Subapplication is the host screen-equivalent ACE generates from a screen image. A single screen image can generate only one Subapplication. However, you can create more than one screen image from a single Screen Definition File. You can also capture and save a single screen many times. Therefore, you can create more than one Subapplication for each host screen, whether from captures or SDFs. However, each Subapplication necessarily represents one host screen.

ACE displays a Subapplication in the form of a window. This window contains several controls, which are graphical equivalents of each host screen element, such as static text, input/output fields etc. A Subapplication is created step-by-step in the **New Subapplication** wizard.

Application

The ACE engine generates a graphical equivalent for each host screen of a host application. The collection of all these graphical equivalents form an Application.

An Application is created step-by-step in the **New Application** wizard.

Library

A library is a number of Subapplications grouped together, with the group having a name. An Application can have many libraries, with each library containing many Subapplications. Libraries are therefore an intermediate level of organization between Applications and Subapplications.

Libraries are logically equivalent to Applications: You can set global properties for a library without affecting the properties of the other libraries.

Libraries are most useful when your host application's screens divide up naturally into categories, either because the screens belong to different modules, or because a number of screens have similar characteristics.

A library is created step-by-step in the **New Library** wizard.

Converting the Host Application

The Host application first undergoes a conversion in ACE so as to properly identify and name the host screens and fields.

To Convert a Host Application Using ACE:

1. Create an Application.
2. Create Screen Images.
3. Create Subapplications.
4. Work through the views to optimize the converted application.
5. Compile the Application (Generate Runtime).

When the host information has been processed, the JIS Application's deployment environment is generated and the processed host screen information—in the form of Subapplications—is converted into Java classes. These classes are then installed on the JIS Server computer.

In ACE you perform the following:

- Create GUI representations of each host screen (Subapplications).
- Design the GUIs to fit your needs.
- Generate runtime.
- Create the distributable.

Creating an Application

Each time you start up ACE, you must establish an Application to work in. Either create a new Application by running the **New Application** wizard or open an Application that you have already created during a previous ACE session.

When you have established your working Application, you can do any following:

- Transfer data from the host to your development platform and create new screen images or maintain existing screen images
- Create new Subapplications from an existing screen image. New Subapplications are automatically converted and then presented for further optimization.
- Open an existing Subapplication.
- Create and run an executable composed of any existing Subapplications of the application.

Creating Screen Images

As a first step in the host application conversion, you must prepare host application data for ACE development. This involves either transferring host files to the PC and/or capturing the screens of the host application in a live connection.

- Convert information about the host application into a form that ACE can read.
- Make the information available to ACE.

These two procedures are performed in different ways, and in different order, depending on the type of host and whether you are using screen definition files or screen captures.

The workflow for creating screen definition files on the iSeries is:

1. Compile DDS files into DDO files on the iSeries using the ACE DDS compiler. See the appendix about the DDS compiler in *webMethods JIS: Getting Started with the Automated Conversion Environment*.
2. Transfer the DDO files to your development platform.
3. Create screen images from the DDO files using the **Create Screen Images** wizard. Use the **Create Screen Images** wizard also to resume work on a DDO file. See “iSeries Applications and Screen Definition Files” on page 41.

The workflow for mainframe screen definition files is:

1. Transfer the BMS/MFS/SDF source files to your development platform as text files.
2. Create screen images from the source files using the **Create Screen Images** wizard. See “Mainframe Applications and Screen Definition Files” on page 45. To resume work on a screen definition file use the **Maintain Screen Images** wizard. See “Maintaining Screen Images” on page 47.

The workflow for mainframe and iSeries screen captures is:

1. Establish an Application in ACE.
2. From the **File** menu select **Create Screen Images**. ACE starts the **Create Screen Images** wizard. The wizard prompts you for the information you need to create images from screen captures. See “Capturing Your Host Application” on page 46.

iSeries Applications and Screen Definition Files

DDS (Data Description Specification) files are iSeries files that describe the screens of the application. iSeries applications that have been developed with DDS have special files that contain the DDS screen information. Code is added to the program that loads the DDS object in runtime.

The DDS information is the precise information needed for understanding the screens. When this information is retrieved it can be used effectively by ACE to create a GUI interface for your application. Thus the DDS files can be an important tool for migrating iSeries applications to GUI.

ACE can convert screens with or without using DDS. This section describes the ACE conversion process with DDS.

Creating Screen Images

From the **File** menu select **Create Screen Images**. ACE starts the **Create Screen Images** wizard.

The wizard prompts you for the information you need to create screen images from DDO files. You will need to know the name and path to the temporary directory where you stored the DDO files.

The Create Screen Images From Compiled DDS Dialog Box

Within the **Create Screen Images** wizard you will need to build screen images from the records contained in the DDO files. You do this within the **Create Screen Images From Compiled DDS** dialog box.

The **Create Screen Images - From Compiled DDS** dialog box has the following appearance:

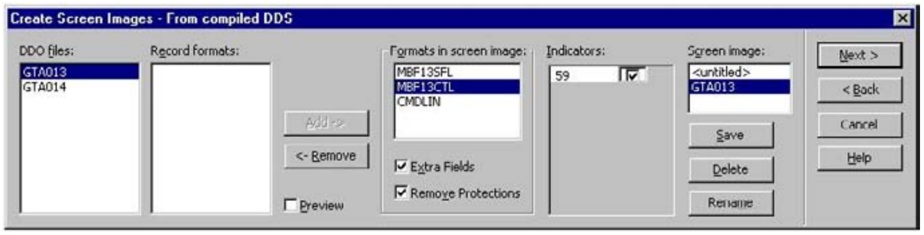


Figure 7. Create screen images dialog box

To create a new screen image or edit an existing screen image:

1 From the **DDO Files** list box select the DDO file from which you wish to define a new screen image or from which a screen image has already been defined.

2 Define a new screen image. Select <untitled> from the **Screen Image** list box.

OR-

Edit an already defined screen image. Select the defined screen image's name from the **Screen Image** list box.

3 To **include** record formats: From the **Record formats** list, select the record formats to be included and click **Add** to transfer them to the **Formats in screen image** list. The screen image displayed in the **Host Session** window is updated to *include* the added record format.

4 To **remove** record formats: From the **Formats in screen image** list select the record formats to be removed and click Remove to return them to the **Record formats** list. The record format name is removed from the **Formats in screen image** list and now appears in the Record formats list. The screen image displayed in the Host Session window is updated to *exclude* the removed record format.

5 From the **Formats in Screen Image** list box select each format one at a time and configure the indicators associated with the record format in the **Indicators** list, where applicable.

6 Click the **Save** button. If you are working on a new screen image, <untitled>, you are prompted to provide the new screen image with a name.

Note: The defined screen images are not actually created until you complete the **Create New Screen Images** wizard.

Features of the Create Screen Images From Compiled DDS Dialog Box

DDO Files   The **DDO files** list contains the names of all the DDO files that are available for processing. These files form the raw material that you use to create screen images. When you select a DDO file from the **DDO files** list, all the screen images already defined from the selected DDO file appear in the **Screen image** list.

The word <untitled> also appears in the **Screen image** list.

Record Formats and Formats in Screen Image

DDO files are composed of smaller units called record formats. A DDO file may contain a single record format or a number of record formats.

- The record formats used in the defined screen image appear in the **Formats in screen image** list. The image created from these records appears in the **Host Session** window.
- The record formats not used in the defined screen image appear in the **Record Formats** list box.

Note: Selecting <untitled> in the **Screen image** list begins the definition of a new screen image. At this point you have not yet added any record formats to the new screen image, so all of the selected DDO file's record formats appear in the **Record formats** list box.

Indicators   Some record formats have multiple versions, controlled by indicators. When you select such a record from the **Formats in screen image** list, the **Indicators** list box lists all the different indicators associated with the record.

Enable or disable an indicator by setting or clearing the check box next to it. The screen image displayed in the **Host Session** window updates to reflect the indicator settings.

Screen Image   The **Screen image** list contains the names of all the screen images defined from the DDO file selected in the **DDO Files** list. To change an existing defined image, select the defined image in the **Screen images** list. To define a new image from the selected DDO file, select <untitled> in the **Screen images** list.

The selected image appears in the **Host Session** window.

Preview   The **Host Session** window displays the screen image defined by the record formats included in the **Formats in screen image** list and the **Indicator** settings. You can preview the effect of transferring a record from the **Record formats** list to the **Formats in screen images** list by setting the **Preview** check box.

When the **Preview** check box is set, the screen image displayed in the **Host Session** window includes the selected record in the **Record formats** list.

The **Preview** feature allows you to quickly see each individual record format's effect on the screen image, since it is faster to select records in the **Record formats** list than it is to transfer records back and forth between the **Record formats** list and the **Formats in screen images** list box.

Save   Click the **Save** button to save the selected configuration of records and indicators as a screen image.

- If you are working on a new screen image—<untitled> is selected in the **Screen image** list—you are prompted to give a name to the new screen image. The new name appears in the **Screen image** list.
- If you are working on a named screen image—the name is selected in the **Screen image** list—then the new image "overwrites" the old image.

Delete   Removes the selected screen image from the **Screen image** list and deletes it from the Application.

Rename   Click **Rename** to give a new name to the selected screen image. The new name appears in the **Screen image** list.

Handling Protected and Hidden Fields   Certain screen images represent two or more screens. If screens generated from a single screen image are used in an Application, ACE must be presented with all the necessary information regarding these screens in order to convert them properly. This entails displaying all extra fields, as well as protected fields.

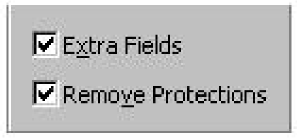

[✓] Extra Fields
[✓] Remove Protections

Figure 8. Handling hidden and protected fields

Extra Fields   The **Extra Fields** check box refers to fields which are either not shown because of indicators or are not initially displayed. These fields are hidden or withheld from a screen. When the **Extra Fields** check box is set, the hidden property of all fields is overridden, and all the fields are displayed.

The display of all fields is necessary for the conversion in ACE, even if not all fields can be displayed when the Application is running. In runtime, ACE refers to the host screen to determine which fields will be presented in the individual screens.

If colors or other keywords are conditioned by indicators that are currently switched off, they are not shown or "forced" by setting the **Extra Fields** check box. The **Extra Fields** check box only forces hidden *fields* to appear.

Remove Protections   The **Remove Protections** check box refers to input fields which are considered “protected.” Protected fields are fields that have received a keyword DSPATR (PR) which indicates that the specified field may not receive input. When the **Remove Protections** check box is set, all protected input fields are displayed as plain input fields, even if this cannot happen when the application is running.

In runtime, ACE refers to the host screen to determine which fields are protected in the individual screens.

Mainframe Applications and Screen Definition Files

BMS (Basic Mapping Support) and MFS (Message Format Service) are two forms of mainframe files that define application screens. They are types of Screen Definition Files. The information contained in them is processed and then used by ACE.

ACE can convert screens with or without using SDF. The following describes the process of creating screen images from BMS or MFS files.

Creating Screen Images

From the **File** menu select **Create Screen Images**. ACE starts the **Create Screen Images** wizard.

The wizard prompts you for the information you need to create screen images from BMS or MFS files. You will need to know the name and path to the temporary directory where you stored the BMS or MFS files.

Screen Image Name Source

The wizard asks you which parameter you want ACE to use as a screen image name.

You can select one of three options:

- Map Name
- Mapset Name
- Field at Position

Note: The field to be used by its location. Enter the XY coordinates for the attribute of the field to be used as the name. The X and Y coordinates start from zero.

If a second screen image has the same name as a previously created screen image, ACE names the new screen image by adding a character to the name. In cases where the name is too long, a character is removed from the name.

Capturing Your Host Application

Where an application was developed without the use of Screen Definition Files or where individual host screens within an application lack Screen Definition Files, you create screen images by capturing screens online from the host application using ACE's screen capture facility. You need to have a live connection to the host, either through TCP/IP, a router or other channels. You may also need to know the address of your host.

Creating Screen Images From Screen Captures

To create screen images from screen captures:

1. Establish an Application in ACE.
2. From the **File** menu select **Create Screen Images**. ACE starts the **Create Screen Images** wizard.

The wizard prompts you for the information you need to create screen images from screen captures.

Combining a Screen Capture and a BMS or MFS File

The host application has the ability to modify the content of fields and their attributes, as defined in a BMS or MFS file.

In some cases, the screen image created by the BMS support in ACE, may lack important information added by the host application. This situation might occur when an output list is filled and divided into columns in runtime, while the BMS or MFS file just defines empty lines without any division into columns whatsoever. Without the screen capture ACE is missing some of the important information. Such a situation can be rectified by combining the screen capture with the BMS or MFS derived screen image.

You can combine the screen image with a screen capture by using the **Edit Screen Image** wizard available in Host View's **Host** menu.

You will need to:

1. Perform the screen capture.
2. Open the BMS or MFS screen image's Subapplication in ACE.
3. Invoke the **Edit Screen Image** wizard to combine the screen capture with the BMS or MFS file. You will need to know the path to the captured screen.

Maintaining Screen Images

When the host screens change as a result of developing the host application, you can update your screen images to reflect these changes. You may have fine tuned the Subapplication created from the original image. The Maintain Screen Images process preserves the fine tuning connected to unchanged parts of the host screen.

To maintain screen images:

- From the **File** menu select **Maintain Screen Images**.

  ACE starts the **Maintain Screen Images** wizard.

The operation of the wizard is essentially the same as for creating screen images. When the wizard prompts you for the required information, you will need to know the name and path to the temporary directory, where you stored the updated DDOs (for iSeries users) or updated BMS or MFS files (for mainframe users).

If new fields were added to the DDS file, then the **Extra Fields** check box should be cleared and set in the wizard's **Create Screen Images From Compiled DDS** step. This re-evaluates and recalculates the fields.

Glossary of File Formats

Table 4 shows the different file formats used in ACE.

Table 4. ACE file formats(Sheet 1 of 2)

Format	Description
MFS	These files contain the source code that defines the screen as transferred from the mainframe. The source code comprises hierarchically arranged units called Mapsets, Maps and Fields.
BMS	These files contain the source code that defines the screen as transferred from the mainframe. The source code comprises hierarchically arranged units called Mapsets, Maps and Fields.
SDF	Processed BMS or MFS files. They contain all the useful information that was extracted from the source screen definition files. ACE uses the information in this form to create SDI and PNL files.

Table 4. ACE file formats(Sheet 2 of 2)

Format	Description
SDI	These files contain all of the BMS information to be used by ACE in the conversion process for a single screen image.
IND	These files provide ACE with the ability to find screen images that were created using a given map or mapset in the **Create Screen Images** dialog box. They contain references from each map to the screen images that were created using the map.
DDS	These files contain the source code that defines the screen as transferred from the iSeries.
DDO	DDS files converted into a format that can be used by ACE. ACE uses this format to create DDI and PNL files.
DDI	These files contain all of the DDO information to be used by ACE for creating a single screen image.
PNL (Panel)	The image of a particular screen. PNL files are either screen captures combined with BMS screen images, files generated by the BMS support or files generated by the DDO support.

Note: For each Mapset, one BMS/MFS, one SDF, and one IND file are produced. For each Screen image, one SDI and one PNL file are produced.

Creating Subapplications

ACE automatically converts host screen images into JIS screen-equivalents. Run this conversion by creating a Subapplication for each screen image:

- When you finish creating a new screen image, the **Create Screen Images** wizard prompts you to create a Subapplication for the screen image.
- At any time, you can create a Subapplication for an existing screen image.

In either case, the **New Subapplication** wizard takes you through the steps of creating the Subapplication.

Optimizing the Converted Application

You optimize the automatic conversion output using the tools available in the different ACE Views. The order in which you work is up to you. Table 5 offers a brief description of each View in a typical working order.

Table 5. ACE views

ACE views	Description
Host View	In Host View you see the original screen image. Examine the contents of the screen image and verify that it is correct.
Layout View	In Layout View you use the sections provided to define fields as Input/Output enabled or to ignore fields you do not require. See the chapter about Layout View operations in *webMethods JIS: Basic User's Guide*.
Design View	In Design View you retrieve or rename field names. This is also where you create or modify methods. See the chapter about subapplication-specific modifications of the design in *webMethods JIS: Basic User's Guide*.
Runtime Screen Identification View	In Runtime Screen Identification View you change, if necessary, the markings that uniquely identify the host screen during runtime. Note that it is imperative that **all** screens be identified during runtime. See the chapter about the Runtime Screen Identification View in *webMethods JIS: Basic User's Guide*.
Analysis View	In Analysis View you change, if desired, ACE's analysis of host screen lists. You need to work in Analysis View only when a Subapplication contains lists. See the chapter about operations performed in the Analysis View in *webMethods JIS: Basic User's Guide*.

Considerations When Working in ACE

Since ACE is used for more than one product, it is important to know which features of ACE do not apply to the XHTML client and, therefore, are not supported by this product. These ACE features are not supported by this product because ACE is used to manufacture different clients, such as a Java client, which uses TCP/IP. Various limitations, therefore, should be taken in to consideration, when working in ACE.

These limitations can be categorized in the following manner:

- HTML limitations
- HTTP limitations
- Browser limitations

HTML Limitations

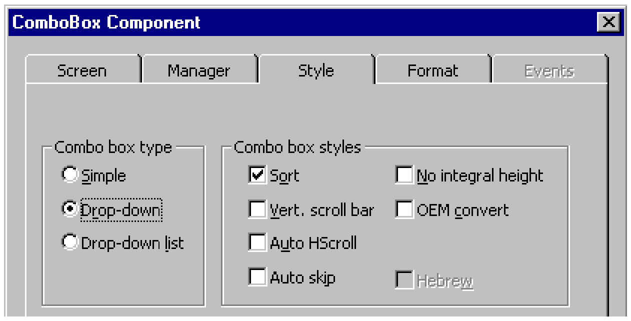
Due to limitations in HTML, the XHTML client cannot fully exploit all features available in Ace.

These features are:

- Combo boxes - drop-down list property not supported
- Check boxes, Radio buttons - control and text treated as separate controls
- Spin controls
- Masking text
- Date controls - based on JavaScript
- Tab controls - based on JavaScript; multi-line tabs not supported

Combo Boxes

HTML does not support the “drop-down list” property of a combo box.



Combo box type
- Simple
- [✓ Drop-down]
- Drop-down list

Combo box styles
- ✓ Sort
- No integral height
- Vert. scroll bar
- OEM convert
- Auto HScroll
- Auto skip
- Hebrew

Figure 9. Combo box style tab

You should, therefore, select one of the other combo box styles.

Check Boxes and Radio Buttons

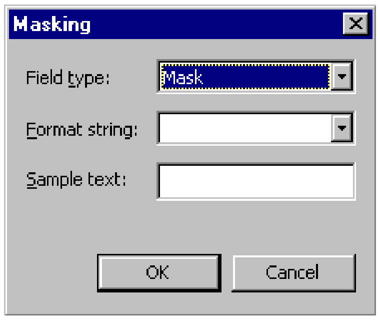
HTML cannot display a check box with its caption as one single control. Instead, they can be displayed as two separate controls. The same applies for radio buttons. Therefore, you must take this into consideration when designing the GUI.

Spin Controls

HTML cannot display Spin controls. Even though it is possible to choose these controls in ACE while designing the GUI, you should not use them. If you do, the XHTML client displays them as normal empty edit boxes with static text.

Masking Text

HTML does not support the ability to mask text. Therefore, when selecting a text format in the **Style** tab, you should not choose the text masking option.



Field type: Mask
Format string:
Sample text:

Figure 10. Masking text dialog box

Date Controls

In ACE, a date control looks as follows:

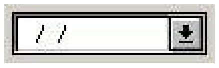


Figure 11. Date control in ACE

In runtime, however, the date control is displayed with a calendar icon instead of the arrow icon

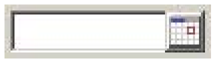


Figure 12. Date control in runtime

You use the calendar icon to open a calendar window. The calendar window, the validity checking and the formatting of the date string from and to the server, are all based on JavaScript. There are two cases in which JavaScript cannot be activated:

1 In browsers that do not support JavaScript - the date control is displayed as a regular text box.
2 In browsers in which JavaScript has been disabled - the date control will look like a Date control, but will not have the functionality of a date control.

Tab Controls

The following limitations apply to Tab controls:

- Tab controls are supported only in browsers in which JavaScript is enabled.
- Multi-line tabs are not supported.

Host Paging from Within Tables.

Provides functionality for Page Up/Down keys to perform host paging from within a table.

There are two scenarios:

- Multi-page table containing multiple pages of information contained within a single table structure.
- Self contained table that does not contain a scroll bar, although the table itself contains more rows that those displayed.

Multi-page table: If the table contains multiple pages, they can be accessed by using the Page Up/Down keys, paging down a table will retrieve the next page to be displayed on the table from the host machine.

Single table without additional pages: Tables that do not have enough display lines, due to space or design restrictions, or do not have a scroll bar can retrieve extra lines present in cache by holding down the Control Key and pressing the Page Up/Down Keys. Thus enabling the user to scroll down the table adding a single line with each key stroke on the Page Up/Down keys.

HTTP Limitations

The nature of HTTP dictates that any change of the screen that is displayed on the client's web browser is the result of a process that was initiated by the client. Therefore, any spontaneous updates from the host may not be sent to the client until the next time that the client initiates a process.

Certain ACE features cannot be fully exploited with a client that is connected via HTTP.

These features are:

- DIL messages
- Message handling
- Message boxes

DIL Messages

Dynamic Information Lines (DIL) are sent from the host during runtime. If the host sends such a spontaneous message while the end-user is on a screen, there is no way that the end-user can receive this message right when it is sent. Instead, as the end-user enters a new screen, the last DIL message is incorporated into the next screen's HTML. DIL messages are displayed at the top or the bottom of the HTML page. This is determined in the JIS Server INI file's `[XHTML]` section by setting the `DILPosition` parameter.

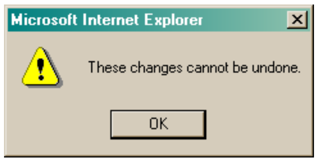
Message Handling

The JIS Server is designed to react towards spontaneous messages sent from the host. However, you cannot set a Message-Handling method to change anything in the Client's display, because, as mentioned earlier, changes to the Client's display are the result of a process initiated by the Client itself.

Message Boxes

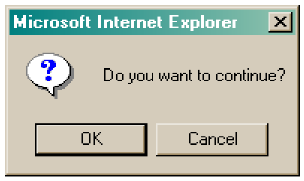
Message boxes can be displayed to the runtime end-user by using the MsgBox method line type in an ACE method. The developer chooses the type of message box in the MsgBox method dialog. Three types of message boxes are supported in the XHTML client: Exclamation, Yes/No, and OK/Cancel.

The Exclamation message box looks like this:


Microsoft Internet Explorer

These changes cannot be undone.

Choosing either the Yes/No message box or the OK/Cancel message box generates an OK/Cancel message box that looks like this:



Do you want to continue?

If the OK/Cancel message box was chosen, the message box returns 1 for OK and 0 for Cancel. If the Yes/No message box type was chosen, the message box returns 7 for Yes and 6 for No.

There is a limitation to the message box feature in the XHTML client in that you cannot set the text value of the title bar. The message text, of course, can be set to any string value.

Browser Limitations

The XHTML client is displayed differently on different browsers. Each browser has its own limitations, which affect the XHTML client differently.

There is also a general browser limitation:

- Accelerator keys such as Ctrl, Alt, Page Up, and so on, are not currently supported in the XHTML client. FKey support is available for keys F1 through F24, provided the FkeySupport parameter is set to 1 (this is the default value) in the XHTML Section of the runtime `INI` file.

Wait for Screen

Note: The following section requires you to have read the chapters relating to Methods in the ACE documentation.

Depending on the design of the host application, it can happen that in the interval between a user action and the display of the resulting screen:

- An intermediate screen or screens are briefly displayed by the host, or
- The desired screen is sent by the host to the terminal in more than one RU (request unit - a bundle of data), giving a flashing appearance to the screen until it has been completely assembled on the host terminal.

These phenomena are undesirable, because you want the end user to see a complete and correct screen, and not an intermediate or partially assembled screen.

The Wait for Screen feature was designed to give the server the opportunity to wait for the desired screen to appear, while ignoring unexpected and/or incomplete screens. The feature consists of two elements:

- A DoMethod called **`SetWaitForScreenState`**.
- A System-Triggered Method called **`UserAcceptScreen`**.

We first explain on a general level how to implement the wait function of the Wait for Screen feature. More detailed explanations are then given about each of its components.

If you want to use the Wait for Screen feature for the first screen of an Application, special considerations apply, which are discussed in "Using Wait for Screen on the First Screen of an Application" on page 61.

General Explanation

During a transaction's execution, behind-the-scenes navigation may occur when an AID key is sent to the host and the JIS Server starts waiting for a response. The most common situation where this may occur is when a method is composed of:

- A HostType line followed by a **`MoveAccordingToHost`** DoMethod line, OR
- A **`SendAIDKeyByString`** DoMethod line followed by a **`MoveAccordingToHost`** DoMethod line.

To implement the waiting function, the **`SetWaitForScreenState`** DoMethod must be added **before** the HostType line or the **`SendAIDKeyByString`** DoMethod line. The parameters which you enter for **`SetWaitForScreenState`** control the behavior of the wait function.

With these parameters, you specify:

- The list of screens that the server should accept or ignore.
- The maximum length of time the server is to wait before it accepts the screen that is currently presented by the host, even if that screen is on the "ignore" list or is not on the "accept" list.

The **`SetWaitForScreenState`** DoMethod with its parameters is presented below in detail.

The Wait function does not directly use the parameters from **`SetWaitForScreenState`**. Instead, it passes them on to the System-Triggered method **`UserAcceptScreen`**. This System-Triggered method is activated immediately after the server receives a screen from the host, before that screen is displayed on the end-user's browser. **`UserAcceptScreen`** decides whether to accept the screen or not, according to the parameters that were set in **`SetWaitForScreenState`**. If the method returns true, the current screen is accepted. If it returns false, the current screen is ignored, i.e., not displayed, and the system waits for an acceptable screen until the time-out limit is reached.

The SetWaitForScreenState DoMethod

The **`SetWaitForScreenState`** DoMethod tells the server how long to wait for the correct screen to be sent from the host.

The correct screen and the wait interval are defined by three parameters listed in Table 6:

Table 6. Parameters in SetWaitForScreenState

Parameter	Description
screenList	This parameter is of String type, so it must be enclosed in quotation marks. This value is usually expressed as a constant consisting of a list of one or more screen names (Subapplication names) separated by semi-colons. The list specifies screens to be accepted, or screens to be ignored. The second parameter, **accept**, sets whether the screens on the list are accepted or ignored. When this parameter is set to a blank value (“”), the list is empty.
accept	This parameter is of Boolean type, i.e., _TRUE or _FALSE. When set to _TRUE, the runtime allows ONLY screens specified in **screenList** to be accepted by the runtime. When set to _FALSE, if any of the screens specified in **screenList** are received from the host by the server, they are ignored.
timeout	This parameter is of Integer type. In this parameter you specify the timeout value in milliseconds. The timeout is the time after which the server accepts the last screen sent by the host, irrespective of whether or not the screen fulfills the conditions imposed by first two parameters. If a time of zero (0) is specified, the wait function is not activated at all, and all screens received from the host are displayed.

The UserAcceptScreen System-Triggered Method

`UserAcceptScreen` is a System-Triggered method that is activated automatically after the host sends a screen to the server, and before that screen is accepted. **`UserAcceptScreen`** returns either true or false. You can modify **`UserAcceptScreen`** if you want to do further processing or examination of the parameters used by **`SetWaitForScreenState`** before deciding whether to accept or ignore a screen. If you have no need for such additional processing, do not modify **`UserAcceptScreen`**.

When the new screen arrives from the host, **`UserAcceptScreen`** searches for the screen's name in the **screenList** parameter list. The method accepts the screen and returns true if:

- The name appears in the list, and the list is of the "accept" type, i.e., the **accept** parameter is set to _TRUE.

  -OR-
- The name does not appear in the list, and the list is of the "ignore" type, i.e., the **accept** parameter is set to _FALSE.

When ***UserAcceptScreen*** returns true, the screen is sent to the user, otherwise, the screen is ignored, the method returns false and the screen is not displayed. The JIS Server keeps waiting until the timeout period has elapsed. When this happens, the method returns true, the server accepts the last screen that arrived from the host and sends it to the end-user.

Note: When the timeout parameter's value is zero, **`UserAcceptScreen`** is not activated. In this case, the screen arriving from the host is immediately sent to the end-user.

Customizing the SetWaitForScreen Function

You can use the **`UserAcceptScreen`** method to customize the behavior of the **`SetWaitForScreenState`** DoMethod. To get to the ***UserAcceptScreen*** method, in Design View, in the **Design** menu select **System Triggered Methods**. **`UserAcceptScreen`** can be modified only at the General Methods level; that is, any changes to it affect all Subapplications in the Application.

If you wish to modify **`UserAcceptScreen`**, use the following guidelines:

- The method's return type is boolean. It must return either true or false, no other values are possible.
- Returning false causes the server to ignore the screen and wait. Returning true accepts the screen immediately.

- When the timeout is over, the method is not activated any more, and the last screen that arrived from the host is sent to the end-user.
- In order to get the **`SetWaitForScreenState`** parameters that are passed to **`UserAcceptScreen`**, use the **`GetCurrentContextParm`** DoMethod. This is explained below.

Using GetCurrentContextParm with UserAcceptScreen

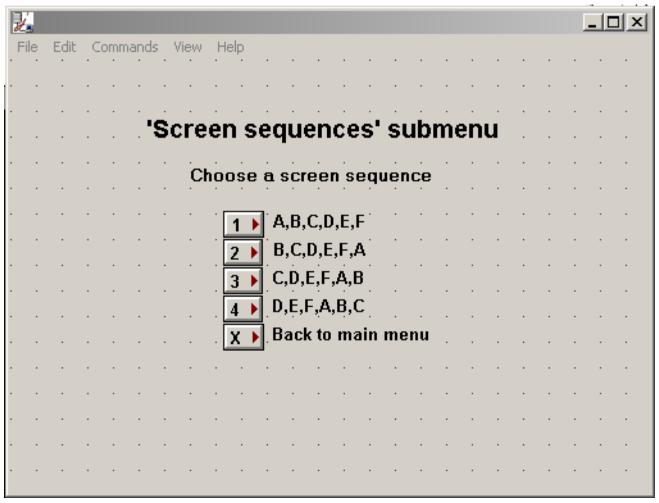
`UserAcceptScreen` receives the **`SetWaitForScreenState`** values as strings contained in the Context Parameter. The **`GetCurrentContextParm`** DoMethod is used to retrieve a string containing these values. **`GetCurrentContextParm`** has one parameter, **`parmNumber`**, which can take one of four values: 0, 1, 2, or 3.

- **`GetCurrentContextParm`** `(0)` returns the name of the screen sent by the host, for example: "`MBF001`".
- **`GetCurrentContextParm`** `(1)` returns the list of screens from the **screenList** parameter.
- **`GetCurrentContextParm`** `(2)` returns the value of the **accept** parameter - "`true`" or "`false`".
- **`GetCurrentContextParm`** `(3)` returns the value of the **timeout** parameter.

Note: When the timeout parameter is set to zero, ***UserAcceptScreen*** is not activated, and modifications to the method are meaningless. The Context Parameter is read only once by the method.

Example of the Use of SetWaitForScreen

This section presents an example of how to use the Wait for Screen feature. The example illustrates the functionality that makes the Wait for Screen feature useful.



'Screen sequences' submenu

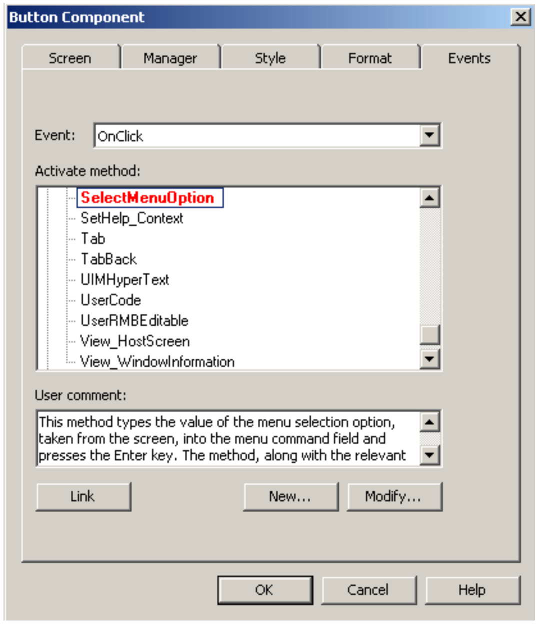
Choose a screen sequence

1 A,B,C,D,E,F
2 B,C,D,E,F,A
3 C,D,E,F,A,B
4 D,E,F,A,B,C
X Back to main menu

Figure 13. A sample subapplication

Figure 13 shows a Sub application designed for testing the Wait for Screen feature. Menu option 1 causes the display of six screens in sequence, screens A, B, C, D, E, and F. In our example, we use the Wait for Screen function to modify the operation of this menu option so that only the last screen, screen F, is displayed.

In ACE's Design View, we double click on the button control for menu option 1 in order to display the **Button Component** dialog box, shown in Figure 14



Event: OnClick

Activate method:

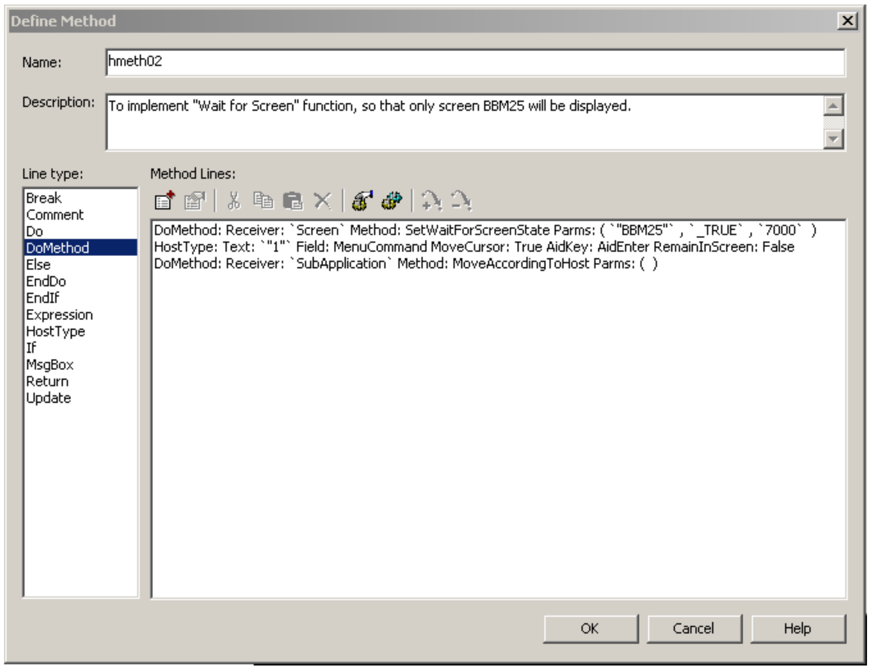
SelectMenuOption
SetHelp_Context
Tab
TabBack
UIMHyperText
UserCode
UserRMBEditable
View_HostScreen
View_WindowInformation

User comment:

This method types the value of the menu selection option, taken from the screen, into the menu command field and presses the Enter key. The method, along with the relevant

Figure 14. The button component dialog box

We see in the Events tab that the **`SelectMenuOption`** method is already attached to this button. It was attached automatically during screen generation. Because we want to add the Wait for Screen function, we will build a new method to be linked to this button in place of **`SelectMenuOption`**. If we added our changes to **`SelectMenuOption`** they would affect all screens, because the **`SelectMenuOption`** method can only be modified globally.



Name: hmeth02

Description: To implement "Wait for Screen" function, so that only screen BBM25 will be displayed.

```
DoMethod: Receiver: `Screen` Method: SetWaitForScreenState Parms: ( `"BBM25"` , `_TRUE` , `7000` )
HostType: Text: `"1"` Field: MenuCommand MoveCursor: True AidKey: AidEnter RemainInScreen: False
DoMethod: Receiver: `SubApplication` Method: MoveAccordingToHost Parms: ( )
```

Figure 15. A method using SetWaitForScreenState

The first line of the new method uses the DoMethod **`SetWaitForScreenState`**. It uses the string “`BBM25`” as the screen name parameter (`BBM25` is the name of the Subapplication for the host screen called “F” on the menu), sets accept to _TRUE, and sets a time limit of seven seconds. In other words, we are telling the server to wait up to seven seconds for screen `BBM25` to appear, and to not display any screens except for `BBM25` before the time limit elapses.

The HostType line in the method types data in the host screen - in this case, a menu option, and sends the AID key “Enter” to the host.

The final line of the method instructs the server to let the GUI window reflect the reactions of the host.

Using Wait for Screen on the First Screen of an Application

`UserAcceptScreen` and **`SetWaitForScreenState`**, as they have been described above, are activated from the current Subapplication. This is a limiting factor in case no Subapplication has yet been entered, a situation typical to the "login" screen, or the first screen of the Application.

To allow the Wait for Screen feature to work in the first screen of the Application, create an `[Initialization]` section in the `<ApplName>.ini` file, and add the parameters of the **`SetWaitForScreenState`** DoMethod to it, as shown in Table 7:

```
[Initialization]
screenList=<list of screens, separated by semi-colons>
accept=0 or 1
timeout=
```

Note: The screen list need not be enclosed in quotation marks. The accept parameter, which is Boolean, should be 0 for false, or 1 for true. **`SetWaitForScreenState`** is called automatically when the Application loads.
Also, for a first screen situation, a customized **`UserAcceptScreen`** method is not called, the default implementation is called instead.

Table 7. Parameters for first use of SetWaitForScreenState

Parameters	Behavior of wait function	Comments

`screenList="MBF001"`
`accept=_TRUE`
`timeout=30000`
	Screen `MBF001` is accepted immediately. Any other screen is ignored, until thirty seconds have passed. After that time, the current screen is accepted.	This is the usual use of the method.

Common Problems

Table 8 lists common problems you may encounter while working with the Wait for Screen feature

Table 8. Troubleshooting of the Wait for Screen feature

Problem	Common Cause	Solution
All screens are accepted.	Timeout value is zero.	Change the timeout to a non-zero value.
A dialog box "server not responding" appears.	The timeout between end-user and JIS Server is too short and interferes with the waiting function timeout.	Increase the timeout value in the HTML setting `CommTimeOut`. The user should click **Yes** to instruct the end-user to keep waiting.
Compilation error in Runtime Generation.	***UserAcceptScreen*** was modified with a wrong return value type, or no return value.	The method must return true or false. No other return value is possible.
Screens continue to appear after the desired screen has arrived.	The host keeps sending screens without waiting.	There is no solution. The normal emulator behavior is to send incoming screens. Modifying UserAcceptScreen may provide a workaround.

Limitations

The DoMethods used to send a key to the host, **`SendASpecificKey`**, **`SendKeyWithoutReset`**, and **`SendKeyByString`** do not work like **`SendAIDKeyByString`**. To use any of these methods, call the DoMethod **`NonBlockingWait`** just after them.

Generating a Runtime

This section describes the compilation process. This process, called Runtime Generation, is the last stage in the conversion process. Use the **Generate Runtime** command in ACE to compile your Application and create the JIS Server.

During runtime generation, Java code that corresponds to each Subapplication is generated. Each Subapplication has its own Java class. Furthermore, static XMLs, CSS files and binary data corresponding to each Subapplication are also generated. The Java classes and static XMLs form part of the JIS Server.

The generated XMLs are then processed and converted to XHTML. These XHTML files are only used as references and will not used by the server.

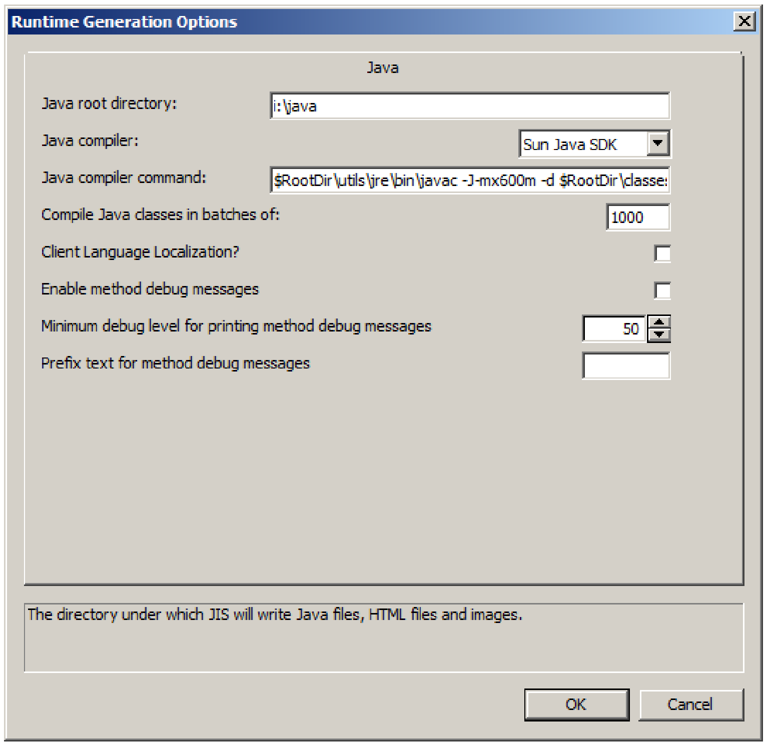
The runtime generation process works in the following manner:

- Generating Java sources and binary data.
- Invoking the Java compiler in order to compile these Java sources into Java class files.
- Composing static XMLs.
- Converting these XMLs to XHTMLs.

Setting Runtime Generation Options

Before generating a runtime, the following options must be defined in ACE:

1 From ACE's **Options** menu, select **Runtime Generation Options**.
The **Runtime Generation Options** dialog box opens:



Java root directory:	j:\java
Java compiler:	Sun Java SDK
Java compiler command:	$RootDir\utils\jre\bin\javac -J-mx600m -d $RootDir\classe
Compile Java classes in batches of:	1000
Client Language Localization?	☐
Enable method debug messages	☐
Minimum debug level for printing method debug messages	50
Prefix text for method debug messages	

The directory under which JIS will write Java files, HTML files and images.

Figure 16. Runtime generation options dialog box

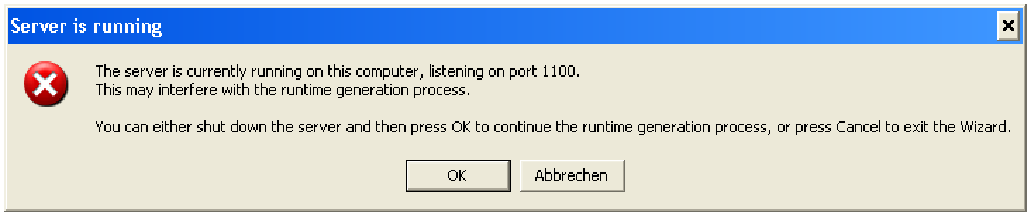
2 Specify the following:

Java root directory	Choose the directory under which the JIS Server files reside.
Java compiler	Choose a Java compiler.
Java compiler command	The default command for the chosen compiler appears. If the Java compiler is not in your path, you can change the command line to specify the full path. The information that is written in the Java compiler command line, is dependent upon the compiler which is used. The Java compiler command default is the Sun Java SDK Compiler. However, if you are using a different compiler, you should change the Java compiler command. You can use the following internal variables in the Java Compiler Command: • **`$RootDir`** At compile time this variable is automatically replaced with the Java root directory that you specified above. • **`$File`** The **`$File`** variable is required. At compile time this variable is automatically replaced with the name of the file to be compiled.
Compile Java classes in batches of	Specify the number of Java classes you wish to be compiled in each invocation of the compiler. Reducing the number of classes compiled at a time reduces the memory consumption of the compiler.
Client Language Localization	Set this check box to enable the Language Localization feature. For a detailed discussion about this feature, refer to Chapter 6 - "Language Localization" on page 227.

Enable method debug messages	When checked, this option causes code for the printing of detailed debug messages to be generated along with the method code. The method debug messages are written to the JIS Server log at runtime. The method debug messages assist in the tracing and debugging of user-written methods. You can use the METHOD log filter to suppress the printing of all other log messages except for the method debug messages. See "Debug Filters" on page 166.
Minimum debug level for printing method debug messages	The debug level at which method debug messages are triggered. See "Enable method debug messages" above.
Prefix text for method debug messages	Allows you to specify a message prefix to be printed at the start of the method debug messages. Default prefix is "METHOD_DEBUG".

How to Generate a Runtime

Before you initiate the Generate Runtime process, make sure that the Server is closed. Trying to compile your application while the JIS Server is running causes the runtime generation process to fail, and the following message appears:


Server is running

The server is currently running on this computer, listening on port 1100.
This may interfere with the runtime generation process.

You can either shut down the server and then press OK to continue the runtime generation process, or press Cancel to exit the Wizard.

Figure 17. Error message during compilation

Following is a description of how to run the runtime generation process and the information you get from the **Generating the Runtime** dialog box.

To generate a runtime:

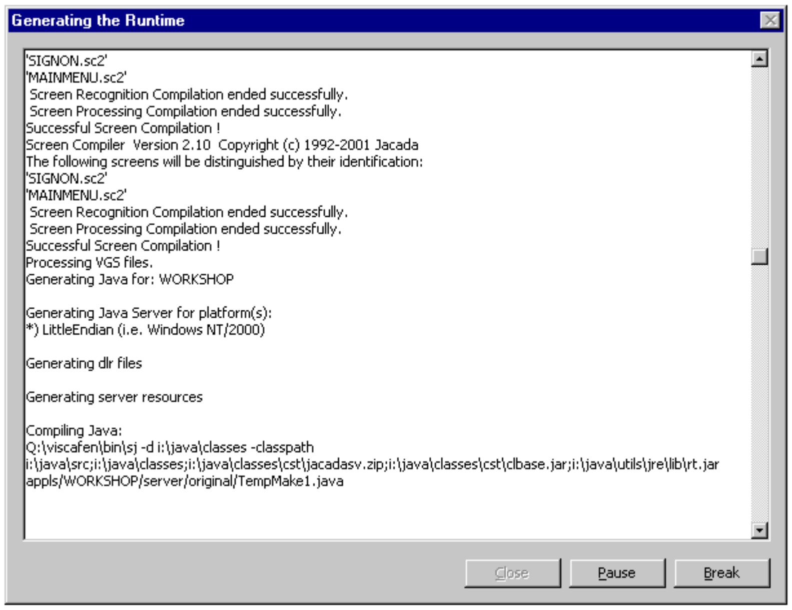
1 From the **File** menu, select **Generate Runtime**. This activates the **Generate Runtime** wizard which generates the JIS Server.

2 In the **Generate Runtime** wizard, choose the platform(s) the Server will run on, such as Windows, OS/400 RISC or Solaris SPARC

3 Continue the wizard to its final step.

The Runtime Generation Process

When generating the runtime, the JIS Server and its associated sources are generated. The Java sources are automatically compiled. Compilation errors or warnings are displayed in the **Generating the Runtime** dialog box.



```
'SIGNON.sc2'
'MAINMENU.sc2'
 Screen Recognition Compilation ended successfully.
 Screen Processing Compilation ended successfully.
Successful Screen Compilation !
Screen Compiler  Version 2.10  Copyright (c) 1992-2001 Jacada
The following screens will be distinguished by their identification:
'SIGNON.sc2'
'MAINMENU.sc2'
 Screen Recognition Compilation ended successfully.
 Screen Processing Compilation ended successfully.
Successful Screen Compilation !
Processing VGS files.
Generating Java for: WORKSHOP

Generating Java Server for platform(s):
*) LittleEndian (i.e. Windows NT/2000)

Generating dlr files

Generating server resources

Compiling Java:
Q:\viscafen\bin\sj -d i:\java\classes -classpath
i:\java\src;i:\java\classes;i:\java\classes\cst\jacadasv.zip;i:\java\classes\cst\clbase.jar;i:\java\utils\jre\lib\rt.jar
appls/WORKSHOP/server/original/TempMake1.java
```

Figure 18. Generating the runtime dialog box

A more detailed log is written to the `makevgs.log` file and `xml.log file` in the ACE root directory.

The Application HTML Files Generated During Compilation

The compilation process generates the following HTML references to the archive files:

<ApplName>-xhtml.html	Runs a client on the JIS proprietary server.
<ApplName>-webapp.html	Runs a client on a J2EE application server. See Chapter 12 - "Application Server Deployment" on page 405 for more information.

Testing the Generated Runtime

After generating a runtime, you must test the JIS Application, before creating the distributable. Some problems in the conversion can only be detected through functional testing.

You test the generated runtime by running the Application on the development machine. In this process, the development machine acts as both XHTML client and JIS Server.

When testing the runtime, you check the following:

- Connectivity with the host.
- Does the JIS Server correctly identify the host screens?
- Each Subapplication's appearance and function in runtime.

Note: The runtime Application appears exactly as it would during full deployment.

There are two ways to run your Application on the development machine:

- Running the Application from within ACE.
- Starting the Application manually.

Activating a Web Server

The JIS Server requires a web server to handle HTTP communication with the client.The default is to use the Jetty HTTP Server, which is an integral part of the JIS Server. Unless specified otherwise, the documentation assumes the Jetty HTTP Server is being used as the runtime Application's web server.

Running the Application from within ACE

To start the Application from within ACE:

1. In ACE, select **File > Run Application**.

   The **Run Application** wizard opens.
2. Follow the wizard's steps to completion. Make sure to specify **XHTML** for **Runtime Type**. As for the rest of the settings, you can work with the default settings.

After completing the wizard, ACE starts the JIS Server and runs the application on your default browser.

Starting the Application Manually

To start the application manually:

1 Activate the JIS Server.
2 Send an HTTP request from your web browser.
3 Test the runtime.

Activating the JIS Server

To activate the **JIS Server**, click the **JIS Server** shortcut icon under your computer's **Start** menu **> Programs > JIS> JIS Server**.

You can also activate the JIS Server by selecting the file `<InstallDir>\jacadasv.bat`

A DOS prompt window opens and after a few seconds the bottom line reads "Server ready: STARTED".

Note: As a result of activating the JIS Server, the Jetty web server is also activated.

Sending an HTTP Request from your Browser

After activating the JIS Server, you can now start the Application. To do this, you write a specific HTTP request in the browser's URL address bar.

The HTTP request should have the following format:

```
http://<IPAddress>:<Port>/<ApplName>-xhtml.html
```

For example:

```
http://localhost:8080/MYAPPL-xhtml.html
```

The HTTP request is comprised of the building blocks listed in Table 9:

Table 9. HTTP request elements (Sheet 1 of 2)

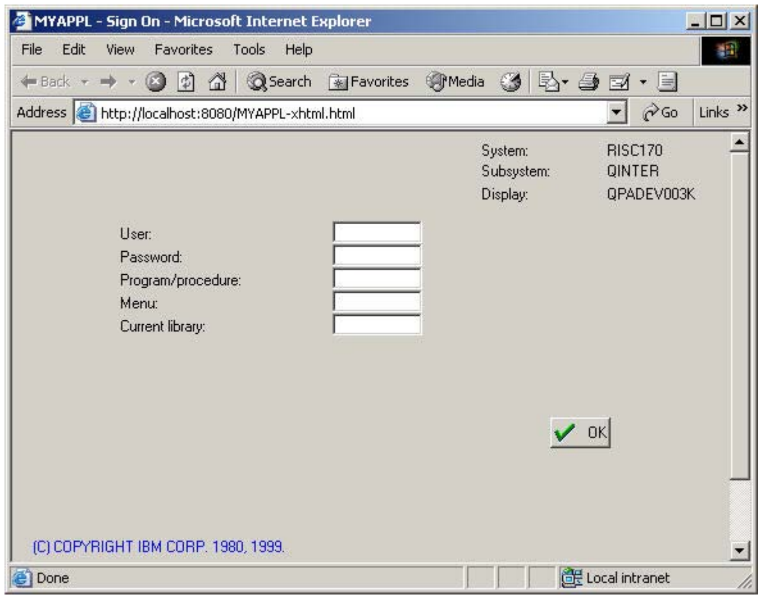
HTTP element	Description
IP Address	The IP address is the JIS Server's IP address. In the above example, the IP address is "`localhost`" because the client is being run from the same machine the JIS Server is installed on.

Table 9. HTTP request elements (Sheet 2 of 2)

HTTP element	Description
Port	The Port is the port number through which the Web Server is set to accept HTTP requests from the Client. The default port number is 8080.
Application HTML	The name of the Application HTML is composed of the ACE Application name followed by the string “-xhtml”. The runtime generation process places this html file in the following directory: `<InstallDir>\JacadaFiles\`

Testing the Runtime

Once the correct HTTP request is entered, the browser should display the corresponding XHTML of the Host Application’s SignOn screen.


MYAPPL - Sign On - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost:8080/MYAPPL-xhtml.html
Go Links

System: RISC170
Subsystem: QINTER
Display: QPADEV003K

User:
Password:
Program/procedure:
Menu:
Current library:

OK
(C) COPYRIGHT IBM CORP. 1980, 1999.
Done
Local intranet

Figure 19. Application’s SignOn HTML page

You may then browse through the host Application. This is done by filling the appropriate fields and clicking the appropriate buttons on the displayed web page.

You should check your Application for each of the following:

- Functionality
- Misidentified Screens

Functionality

Testing a window for functionality means making sure that:

- All controls have functionality attached to them.
- Each control executes the functionality assigned to it. For example, verify that clicking a certain button in the window causes the correct key on the host to be pressed.

Misidentified Screens

When ACE encounters an inconsistency between the host screen and the screen image, the runtime does not display the corresponding GUI. Either a simple GUI is presented or the host screen bleeds through depending on whether or not the Just-in-Time GUI feature was used.

Table 10 will help you track down the reason for the misidentified screens:

Table 10. Troubleshooting misidentified screens

Problem	Solution
An unconverted screen	Create a new Subapplication.
A screen that changed on the host after conversion	Take the updated screen image through the **Maintain Screen Image** wizard and then to Runtime Screen Identification View to update identification.
Incorrect coloring in Runtime Screen Identification	Identify the fields that are colored incorrectly, by using the **Compare to Captured Screen** utility, and then change their coloring.

Packing and Unpacking Applications

Pack/Unpack allows you to create packages of Applications, and to unpack them on your computer or on other computers. For easy transfer, you can define the package to consist of one file, or of a number of files whose size you can set.

For example, Pack/Unpack is useful in the following situations:

- When several developers work on different parts of the same Application.
- When you want to send part or all of your Application to customer support.
- For backup purposes.

In a situation where one central developer works on Application-level elements such as methods, formatting dictionaries, INI files and KnowledgeBases, and other developers work on libraries belonging to the Application, the main use of Pack/Unpack is easy transfer of Applications and libraries between the central and the other developers.

The need for Application/library transfer generally arises when the database files on the machine of a library developer require additions or fine tuning. The library developer then transfers the library to the central developer. The central developer modifies the Application-level elements as needed, and transfers the updated development environment to the library developer.

This ensures constant updating of the development environment and efficient work assignment amongst developers.

For detailed information on packing applications, refer to the Packaging Applications chapter in *webMethods JIS: Advanced Topics*.

Section 508 Guidelines

Section 508 is the US Federal regulation that defines the standard that all computer applications in government offices should accommodate for people with disabilities. The resulting XHTML runtime application consists of HTML pages that meet these guidelines. Each HTML page represents a SubApplication in ACE. Since developers can and should modify SubApplications in ACE, there are a few recommendations and guidelines to follow while working in ACE, to ensure that the resulting HTML page complies with Section 508 guidelines. The guidelines are as follows:

General Guidelines

- **WCAG Guideline 14**
  Use simple and clear language throughout the Application.

- **WCAG Guideline 14**

  Use consistent presentation style throughout the Application.
- **1194.22 (d), WCAG Guideline 6**

  Tabbing order and text flow should be reasonable, and should accurately represent the flow of controls on the screen.

Colors and Graphics

- **1194.22 (a), WCAG Guideline 1**

  Make sure to provide any non-text content with a text equivalent. Make sure no information is conveyed solely in graphic format.
- Make sure that all information available with colors is also available without colors.
- **1194.22 (c), WCAG Guideline 2**

  Make sure background and foreground color combinations provide sufficient contrast.
- **1194.22 (j), WCAG Guideline 7**

  Application pages and animated controls should not flicker at a frequency greater than 2Hz and lower than 55Hz.

Controls

- **1194.22 (n), WCAG Guideline 9**

  Group related controls together, using a Frame or a GroupBox.
- **1194.22 (n), WCAG Guideline 12**

  Make sure that any input control, such as an edit, prompt or combo box, has the correct text before it. The static text usually appears left of the control or above the control. You must make sure that the static text's position in the tabbing order is one less than the control it describes. This static text becomes the control's label, for screen reading purposes.

> Note: This is not necessary for check box and RadioButton controls, since these have their own text as part of the control.

- **1194.22 (n), WCAG Guideline 9**

  It is recommended to add accelerators to controls. This can be done through ACE as follows:
  - For controls that contain their own text, such as buttons and check boxes, insert an ampersand (&) character in the control’s text, before the letter to be used as the accelerator.

- For controls that do NOT contain their own text, such as text boxes and combo boxes, you insert the ampersand (&) character into the text of the static control, that is attached to this control. Add the ampersand (&) character before the letter to be used as the accelerator.

> Note: In the attached static controls' **Style** tab, make sure that the **No Prefix** check box is NOT checked!

- User Extension controls are placed before or after the original Subapplication controls. Regardless of their position on the screen, the order of controls in the HTML source should always be reasonable. Therefore, when writing a User Extension, you must take into consideration how the extension is merged with the original page, and the order of controls in the merged XHTML. To place extension controls AFTER original controls, do the following:
  - In the extension file, insert an empty `<Form>` tag named `jacadaform`.
  - After the empty `<Form>` tag, insert a new `<Form>` tag with the extension's controls.

  This ensures that the extension controls appears after the original controls.
- **1194.22 (a), WCAG Guideline 1**

  Each Button control defined in ACE must have a text description. A Button control's text description is defined in one of the following ways:
  - The contents of the Button control's Balloon Help text.
  - If no Balloon Help text is provided, then the button text is used as the Button control's description.
  - If neither Balloon Help text nor button text are defined in ACE, then the name of the button's image is used as the Button control's description.
- **1194.22 (n), WCAG Guideline 10**

  It is recommended to use default place-holding characters in Subapplications. You add place-holding characters in Design View.

ASCII Art

1194.22 (a), WCAG Guideline 1

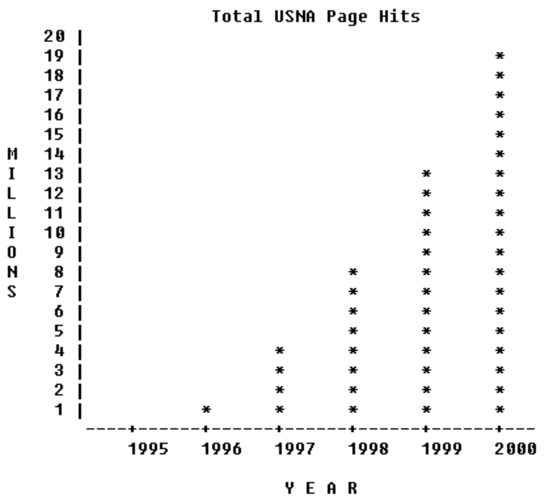
ASCII art refers to text characters and symbols that are combined to create an image. These constructions can be a simple emoticon, such as a smiley ":-)" or something complex, such as a graph. In either case, a person using a screen reader will find it very difficult to understand what you are attempting to communicate, since screen readers interpret ASCII art as a series of independent characters that do not convey any intended message.

Therefore:

- Avoid using any form of ASCII art. If any form of ASCII art, such as “===>”, exists on the original host screen, you must remove it in ACE.
- If a host screen contains a vital piece of information in ASCII art, then replace it with an image with attached descriptive text, which describes exactly what the ASCII art is displaying.

Example:

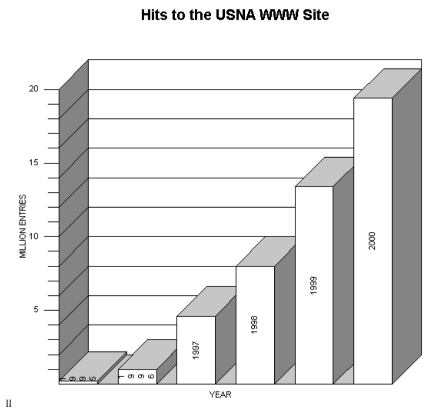
A host screen contains the following ASCII chart, which describes the number of entries to a web page per year during the years 1995 through 2000.



```
                   Total USNA Page Hits
   20 |
   19 |                                       *
   18 |                                       *
   17 |                                       *
   16 |                                       *
   15 |                                       *
M  14 |                                       *
I  13 |                                *      *
L  12 |                                *      *
L  11 |                                *      *
I  10 |                                *      *
O   9 |                                *      *
N   8 |                         *      *      *
S   7 |                         *      *      *
    6 |                         *      *      *
    5 |                         *      *      *
    4 |                  *      *      *      *
    3 |                  *      *      *      *
    2 |                  *      *      *      *
    1 |           *      *      *      *      *
       ----+------+------+------+------+------+---
           1995   1996   1997   1998   1999   2000

                          Y E A R
```

Figure 20. ASCII image

You should remove the ASCII chart and replace it with an image accompanied by descriptive text. A possible replacement image and its descriptive text are:



```
This Chart is entitled "Hits to
the USNA WWW Site". The chart
has a
Y-axis of "Millions of Hits" and
an X-axis of "Years".

The Chart contains the following
data :

1995         255,240
1996         1,011,612
1997         4,882,279
1998         8,060,021
1999        13,580,689
2000        19,335,682
```

Tables

1194.22 (a), WCAG Guideline 5

- Tables contain a Summary option. ACE does not assign a default value for the Summary. You may fill in this option.
- Folded tables are not recommended. Folded tables are represented as single-column tables. Most screen readers read the information from single-column tables, but do not regard these as tables. As a result, some table information is lost. To avoid folded tables in ACE, make sure that the **Fold columns** check box in the table's **Style** tab is NOT checked.

Chapter 4. Deploying the JIS Runtime Application

In order to deploy the JIS runtime Application once it is fully developed, a distributable runtime installation must be created. The runtime environment, contained in this distributable, is then installed on the Server machine that will be running the JIS Application.

This chapter deals with installation aspects of the JIS Server and its associated classes on the platform that will run them.

The following topics are discussed:

- "The JIS Runtime" on page 77
- "The JIS Runtime on Windows" on page 79
- "The JIS Runtime on iSeries" on page 80
- "The JIS Runtime on Solaris" on page 92
- "The JIS Server Command Line Parameters" on page 103

Making the JIS runtime operational on these machines includes creating a JIS Server runtime installation, installing the runtime on the Server machine, and activating the JIS Server.

> **Note:** This chapter assumes that the Jetty HTTP Server, which is an integral part of the JIS Server, is being used as the runtime application's web server. When using an external web server, you must map the web server root directory to the directory under which the runtime is installed.

The JIS Runtime

Once you have successfully created the JIS runtime, you are now ready to install the JIS runtime on the Server machine and run it.

To make a JIS runtime operational you must:

1. Create a runtime installation.
2. Install the runtime.
3. Activate the JIS Server on the Server machine.

The following sections contain background information about the webMethods JIS runtime architecture, and detailed instructions on how to make the JIS runtime operational on Windows, iSeries, and Solaris.

Creating the Runtime Installation

The runtime installation process involves copying the runtime environment from the computer ACE runs on to the computer acting as the Server computer. The first step in this process is to prepare the runtime environment for installation by packaging it. This task is achieved using the **Create Runtime Installation** wizard.

Installing Your Runtime

The JIS Server application classes that were generated by the runtime generation process must be installed on the machine you have designated as your Server machine. This machine can be a PC running Windows, an iSeries, or a Solaris.

Note: Starting with release 9.0 of the webMethods JIS, your runtime can be also deployed to a J2EE application server. The instructions for creating and installing a runtime for a J2EE environment are covered separately, in Chapter 12 - "Application Server Deployment" on page 405.

Keyboard Emulation

By default, the webMethods JIS runtime interprets certain keystrokes and keystroke combinations in special ways, as shown in Table 11.

Table 11. Keyboard emulation

Key seen by runtime	Key sent to host
F1 through F12	PF1 through PF12
Shift+F1 through Shift+F12	PF13 through PF24
Ctrl+Z	Reset
Esc	Attn

To enable the support of Fkeys you must set on the `FkeySupport` flag in the XHTML section of the `jacadasv.ini` file.

The JIS Runtime on Windows

This section describes how to install and operate the webMethods JIS runtime on a Windows Server.

Creating a Runtime Installation

To create a runtime installation:

1. In ACE, from the **Utility** menu choose **Create Runtime Installation**.

   This opens the **Create Runtime Installation** wizard.
2. In the **Create Runtime Installation** wizard choose the following settings:
   - **Deployment Type.** For deployment under the proprietary JIS Server, choose JIS Server Deployment. For deployment to an application server or servlet engine, see Chapter 12 - "Application Server Deployment" on page 405.
   - **Automatic or manual packaging**. It is recommended that you use the Wise Installation Studio. If you choose to use WISE, then the **Create Runtime Installation** wizard produces an information file that WISE can read. If you do not choose WISE, then the **Create Runtime Installation** wizard produces a text file listing the directory structure and files that make up a working runtime on an end-user system.
   - If you are using WISE then you can choose a bitmap to be displayed during the installation on the end user's system.
   - For **Runtime Type**, choose XHTML.
   - For the *JIS Server Platform(s)* option, choose **Windows**.
3. Continue the wizard's steps to the end. At the end of the process ACE creates the following files:
   - **setupjav.exe—**This program installs the JIS Server for your runtime Application.
   - **setup.txt—**if you choose to install the runtime without using the Wise installation program, this text file lists the files that should be included in the runtime Application. This text file also includes an indication of the precise place in the Application runtime directory in which each file appears.

   These files are placed under the ACE root directory, under the directory:

   `<InstallDir>\appls\<ApplName>\install\javasrvr`

   The entire runtime environment is contained in the `setupJav.exe` file. To distribute the webMethods JIS runtime, copy the `setupJav.exe` file into a distributable media.

Installing Your Runtime on Windows

Run the setupJav.exe file (the installation wizard for the webMethods JIS runtime) to automatically install the runtime on Windows. All you have to do is provide the installation wizard with the location to which the runtime is to be installed. The installation wizard installs the JIS Server for your runtime application and creates the JIS Server icons.

To activate the runtime installation wizard:

- Double click the `setupJav.exe` file, in the following directory:

  `<InstallDir>\appls\<ApplName>\install\javasrvr\setupJav.exe`

  If you choose to install the runtime without using the installation wizard, follow the instructions given in the **setup.txt** file. You can install the necessary runtime application files on your Server Computer manually, or using a software-installing utility program.

The JIS Runtime on iSeries

In this section, learn how to install and operate the webMethods JIS runtime on an iSeries.

Creating a Runtime Installation

To create a runtime installation:

1. In ACE, from the **Utility** menu, choose **Create Runtime Installation**. This opens the **Create Runtime Installation** wizard.
2. In the **Create Runtime Installation** wizard choose the following settings:
   - **Automatic or manual packaging**. It is recommended that you use the WISE Installation Studio. If you choose to use WISE, then the **Create Runtime Installation** wizard produces an information file that WISE can read. If you do not choose WISE, then the **Create Runtime Installation** wizard produces a text file listing the directory structure and files that make up a working runtime on an end-user system.
   - If you are using WISE then you can choose a bitmap to be displayed during the installation on the end user's system.
   - For **Runtime Type**, choose XHTML.
   - For the *JIS Server Platform(s)* option, choose **AS/400 (RISC)**.
3. Continue the wizard's steps to the end.

   At the end of the process ACE creates the following files:
   - **setupjav.exe**—This program will install the JIS Server for your runtime application.

- **setup.txt—**if you install the runtime without using the Wise installation program, this text file lists the files that should be included in the runtime application. This text file also includes an indication of the precise place in the application runtime directory in which each file should appear.
- **finst400.txt—**contains a list of records. The iSeries installation uses these records as parameters to install the JIS Server.

These files are placed under the ACE root directory, under the directory:

`<InstallDir>\appls\<ApplName>\install\javasrvr`

The entire runtime environment is contained in the `setupJav.exe` file. To distribute the XHTML Client runtime, copy the `setupJav.exe` file into a distributable media.

Installation Files to be Copied on an iSeries

When installing the runtime on an iSeries, installation files must be copied from the conversion machine and included with the runtime environment. These files are contained within the `Jbs` directory which sits under the ACE root directory:

`<InstallDir>\Install\AS400\Jbs\`

Note: Make sure that these files are included on the distributable media containing the runtime environment.

Installing the Runtime on the iSeries

The iSeries Integrated File System (IFS) enables you to view and manipulate the iSeries's directory and file structure from the PC. Map a local drive on your PC to the iSeries. Then transfer the runtime environment from the PC to the iSeries.

The following sequence and short description that follows outline the steps for installing the runtime on the iSeries. A detailed discussion of each step can be found in the subsequent sections.

To install the runtime on the iSeries:

1. Map the PC to the iSeries IFS.
2. Transfer the runtime environment to the iSeries.
3. Transfer the Server Package to the iSeries.

Mapping the PC to the iSeries IFS enables you to carry out part of the installation process on the iSeries via a PC (step 1 above). If you are unable to map a network drive to the iSeries, you must manually transfer the directories and files that make up the webMethods JIS runtime environment to the iSeries. Both methods are described below.

The installation process itself is performed in two steps. First, you transfer runtime Java classes to the iSeries (step 2 above). This step is wizard driven. Then, you transfer the JIS Server to the iSeries and create libraries on the iSeries to accommodate the runtime environment (step 3 above).

Mapping the PC to the iSeries IFS

If you are unable to map a network drive to the iSeries, skip this section and continue with "Transferring the Runtime Environment to the iSeries Manually" on page 86.

Use a drive sharing mechanism such as Client Access to map a local drive on your PC to the iSeries. Opening the mapped drive on your PC then provides you with direct access to the iSeries's directory and file structure.

There are two ways to map a Network Drive to the iSeries Machine: You can map the drive using Network Neighborhood, or with Windows Explorer.

To map a PC drive to the iSeries machine using Network Neighborhood:

1 From the Network Neighborhood, find the name of your iSeries and double-click on it.

2 Locate the "Shared directory" on the iSeries which represents the "root" of the IFS. This directory is typically labeled "home". If a "home" directory does not exist, consult the iSeries administrator to locate the correct directory.

Example:

```
\\<host name>\home\
```

Click on the "shared directory" with the right mouse button.

3 From the shortcut menu, choose **Map Network Drive**.

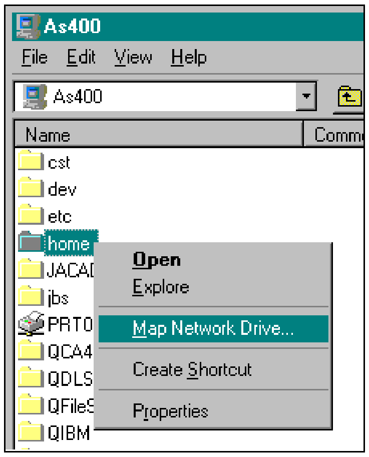


Figure 21. Map network drive

4 In the **Map Network Drive** dialog box, choose a local drive.

   For example: **S:**

5 Create a folder on the newly mapped drive to house the runtime environment.

   For example, create a folder called **jis**.

   Following the previous example, the mapped directory path is: `S:\jis`

   The corresponding iSeries directory path is `/home/jis`.

   Take note to record the iSeries directory path. You will need to enter it in the webMethods JIS installation wizard.

> Note: The runtime environment files must be transferred to the iSeries in their binary form. Therefore, make sure that the mechanism you use for transferring to the iSeries does not translate files from ASCII to EBCDIC.

To map a PC drive to the iSeries machine using Windows Explorer:

1 From the **Tools** menu in Windows Explorer, click on **Map Network Drive**. The **Map Network Drive** dialog box appears.
2 Windows offers you the first available network drive. This is fine, unless for some reason you want to assign a specific drive letter.
3 Choose the “Shared directory” which represents the “root” of the IFS. Typically, this directory is labeled “home”. If you are not sure what directory to use, consult the iSeries administrator.

   If the desired path does not appear in the dropdown list you can type it manually into the Path field.

Example:

```
\\12.34.56.78\home
```

or, with a DNS name

```
\\OURAS400\home
```

4 Click **OK**.
5 Create a folder on the newly mapped drive to house the runtime environment, if such a folder has not already been prepared for you.

Transferring the Runtime Environment to the iSeries via a Mapped Network Drive

The first step in the installation process involves transferring the runtime environment to the iSeries. This step is wizard-driven.

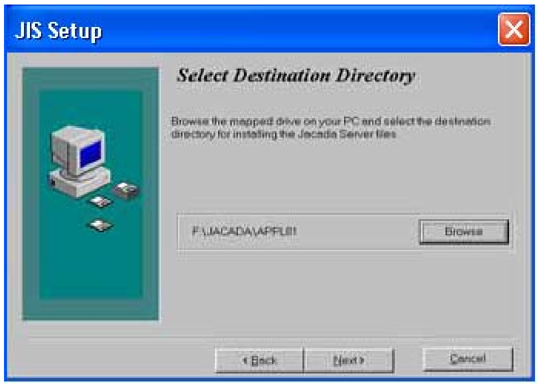
To transfer the runtime environment to the iSeries via a network drive:

1 From the PC, run the webMethods JIS runtime executable by double-clicking on the `setupJav.exe` file, in the following directory:
`<InstallDir>\appls\<ApplName>\install\javasrvr\setupJav.exe`

This invokes the webMethods JIS installation wizard.

2 Enter the User information and click **Next**.

3 The wizard prompts you with the **Select Destination Directory** dialog box:


JIS Setup

Select Destination Directory

Browse the mapped drive on your PC and select the destination directory for installing the Jacada Server files

F:\JACADA\APPL01

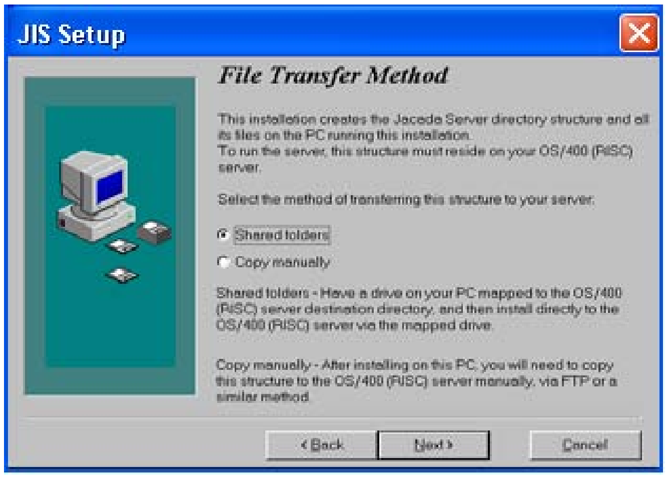
Figure 22. RT transfer to the iSeries via mapped network drive

In the **Select Destination Directory** dialog box, provide a destination directory and click **Next**.

Since the runtime is installed on the IFS on the iSeries, the previously-mapped drive is used to specify the destination directory. As shown above, following the example presented earlier, the destination directory is **F:\jis**.

Note: The **JIS Server Installation** wizard automatically appends the Application name to the runtime root directory. The Application name can be deleted. If you choose to leave it, it will become part of the root directory.

4 In this step, the wizard asks if you are transferring the webMethods JIS file structure to the iSeries via a shared network drive or if you are doing the job manually.



JIS Setup

File Transfer Method

This installation creates the Jacada Server directory structure and all its files on the PC running this installation.
To run the server, this structure must reside on your OS/400 (RISC) server.

Select the method of transferring this structure to your server:

(● Shared folders)
(Copy manually)

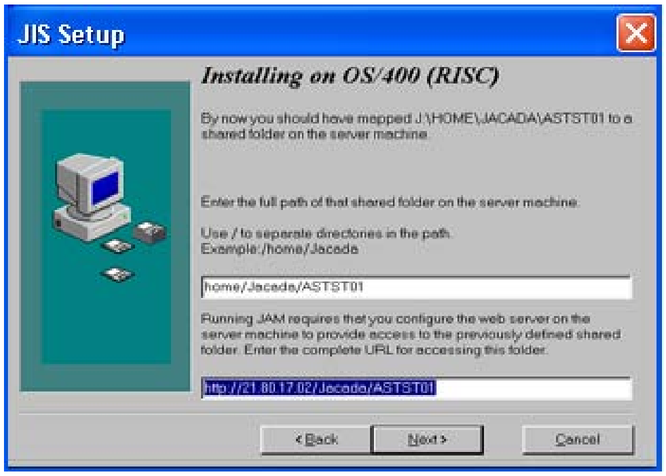
Shared folders - Have a drive on your PC mapped to the OS/400 (RISC) server destination directory, and then install directly to the OS/400 (RISC) server via the mapped drive.

Copy manually - After installing on this PC, you will need to copy this structure to the OS/400 (RISC) server manually, via FTP or a similar method.

Figure 23. RT transfer to the iSeries via mapped network drive (next)

In the **File Transfer Method** dialog box, choose **Shared folders**.

5 The wizard then opens the **Installing on iSeries** dialog box:



JIS Setup

Installing on OS/400 (RISC)

By now you should have mapped J:\HOME\JACADA\ASTST01 to a shared folder on the server machine.

Enter the full path of that shared folder on the server machine.

Use / to separate directories in the path.
Example:/home/Jacada

home/Jacada/ASTST01

Running JAM requires that you configure the web server on the server machine to provide access to the previously defined shared folder. Enter the complete URL for accessing this folder.

http://21.80.17.02/Jacada/ASTST01

Figure 24. RT transfer to the iSeries via mapped network drive (next)

In the **Installing on AS/400** dialog box, set the following options:

Option	Description
Full path to shared folder on runtime machine	The path on the iSeries under which the runtime will be installed. Using the example from above, the path would be **/home/jis**.
Full iSeries directory path	A URL including the iSeries address followed by the directory to which you are transferring the runtime environment.

6 Click **Next**. This initiates the automatic installation of the JIS Server and the Client for your runtime Application. At the end of this process the XHTML Client is transferred to the iSeries.

This completes the first part of the three-part installation process. The second part of the process involves the transfer of the JIS Server from the PC to the iSeries.

Note: Skip the next section, "Transferring the Runtime Environment to the iSeries Manually", and continue with "Transferring the Server Package to the iSeries" on page 87.

Transferring the Runtime Environment to the iSeries Manually

If you were unable to transfer the webMethods JIS runtime environment to the iSeries via a shared network drive, you must do the transfer manually. Otherwise, skip this section and continue with "Transferring the Server Package to the iSeries" on page 87. This process is partially wizard-driven.

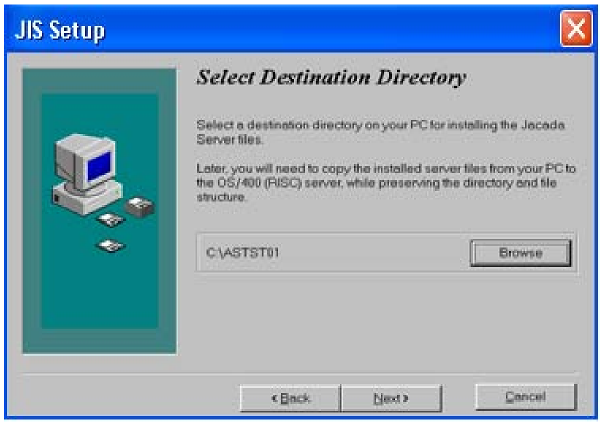
To transfer the runtime environment to the iSeries manually:

1 This step invokes the webMethods JIS installation wizard.

From the PC, run the webMethods JIS runtime executable by double-clicking on the `setupJav.exe` file, in the following directory:
`<InstallDir>\appls\<ApplName>\install\javasrvr\`

2 Enter the User information and click **Next**.

3 The wizard prompts you with the **Select Destination Directory** dialog box box:



Select Destination Directory

Select a destination directory on your PC for installing the Jacada Server files.

Later, you will need to copy the installed server files from your PC to the OS/400 (RISC) server, while preserving the directory and file structure.

C:\ASTST01

Figure 25. Manual RT transfer to the iSeries

In the **Select Destination Directory** dialog box, select a new or empty destination directory on your PC for installing the webMethods JIS runtime files.

4 Continue the wizard to the end.

5 Using `PKZIP`, `FTP`, `PKUNZIP`—or similar utilities—copy the installed runtime files from your PC to the iSeries machine, preserving the directory structure.

Transferring the Server Package to the iSeries

The second part of the installation process uses a batch program to move the JIS Server installation files to the iSeries, in preparation for installing the JIS Server there. You need the IP address of the iSeries along with a valid iSeries login name and password. The batch program copies the `JSERVER` file to the `QGPL` library on the iSeries and restores the installation program to the same library.

To execute the batch program:

1 From the *Windows* **Start** menu, select **Run**.

2 Type `command` and press **OK**. This opens up a command window.

3 In the command window, type `cd <your ACE path>\install\as400\jbs`

Example: `cd c:\jis\install\as400\jbs`

4 Type `jsinstall`

5 You are prompted for the following parameters:

iSeries IP Address	The address of the iSeries machine where the Server is to be installed.
User	Your iSeries login name.

Password	Your iSeries password.

The batch program copies the save file `JSERVER` to the iSeries library called `QGPL`, and restores the installation program to the same library.

Running the Install Program on the iSeries

After executing `JSINSTALL` on the PC as described in the previous section, you have an installation program, also called `JSINSTALL`, in the `QGPL` library on the iSeries.

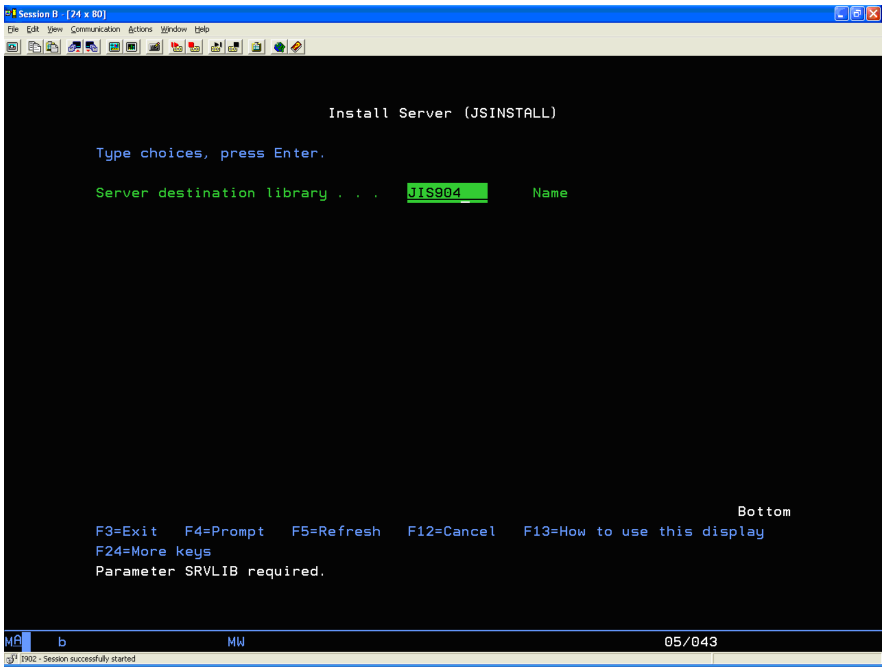
To run the installation program on the iSeries:

1 Establish a connection to the iSeries and logon.

2 Make sure `QGPL` is in your library list (it should be there by default).

3 Type `JSINSTALL` and press **F4**.

The following screen is presented:

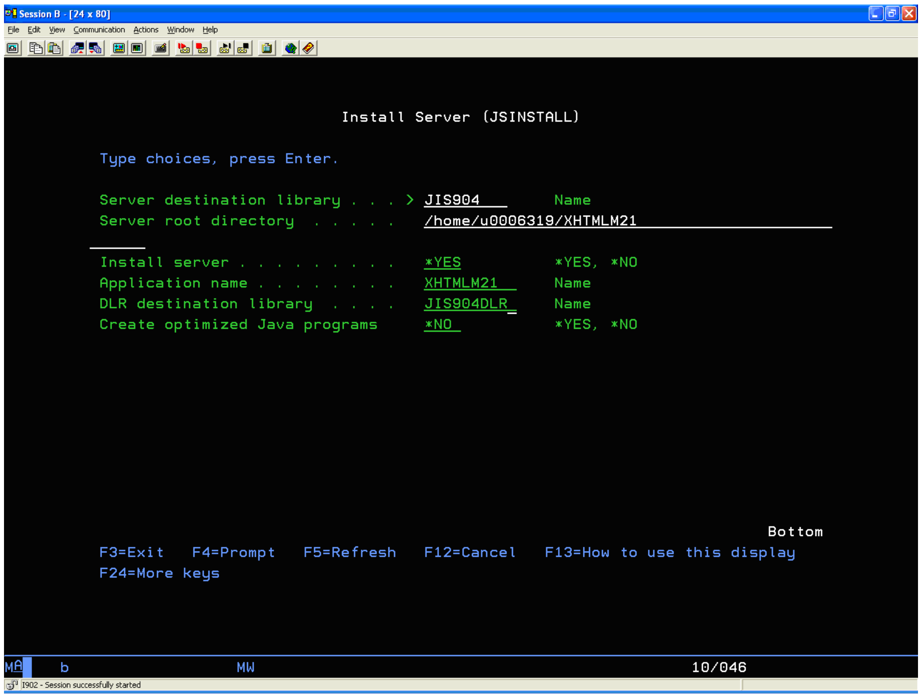


```
                          Install Server (JSINSTALL)

 Type choices, press Enter.

 Server destination library . . .   JIS904        Name









                                                                        Bottom
 F3=Exit   F4=Prompt   F5=Refresh   F12=Cancel   F13=How to use this display
 F24=More keys
 Parameter SRVLIB required.

MA   b                 MW                                    05/043
I902 - Session successfully started
```

Figure 26. JIS Server installation on the iSeries

4 Fill in the name of the library where you want the Server to be installed. If the library you specify does not exist, it is created.

Several additional parameter fields are then displayed:



```
                         Install Server (JSINSTALL)

 Type choices, press Enter.

 Server destination library . . . > JIS904        Name
 Server root directory  . . . . .   /home/u0006319/XHTMLM21

 Install server . . . . . . . . .   *YES          *YES, *NO
 Application name . . . . . . . .   XHTMLM21      Name
 DLR destination library  . . . .   JIS904DLR     Name
 Create optimized Java programs     *NO           *YES, *NO

                                                                       Bottom
 F3=Exit   F4=Prompt   F5=Refresh   F12=Cancel   F13=How to use this display
 F24=More keys

MA     b                  MW                                   10/046
I902 - Session successfully started
```

Figure 27. JIS Server installation on the iSeries: options

5 Fill in the fields according to the following instructions:

Field	Instructions
JIS Server root directory	This is the directory you specified above, in the screen “Installing on OS/400 (RISC)” in the section “Transferring the Runtime Environment to the iSeries”. If the library specified in ‘Server destination library’ already contains a Server, the root directory of that Server appears as default.
Install Server	Type `*YES` if you want the Server to be [re]installed. Type `*NO` if you want to install only the Application. If the destination library already contains a Server, the default is `*NO`.
Application name	The name of the Application being installed with the Server.
DLR destination library	During the installation, certain files—called DLR files—are saved to the library designated here. If the library name you specify does not exist, it is created.
Create optimized Java program	This parameter indicates whether the Java programs are recompiled and optimized for better performance. Type `*YES` or `*NO`. This step is technically optional, but it is **recommended**. Performing the compilations at this time eliminates the need for the iSeries to compile Java objects from Java classes each time they are invoked by the client. By compiling the objects now, application runtime performance is considerably improved. Depending on the capacity of the iSeries, the size of the application, and the optimization level you select, compiling the Java objects now may add up to **several hours** to the install process.

6 After filling in the parameters, press **Enter** to start the installation. The installation process is activated.

The system issues messages to help you follow the progress of the installation procedure. In case of a problem, an error message is displayed.

Activating the JIS Server on the iSeries

You activate the JIS Server by typing a command in the iSeries command line.

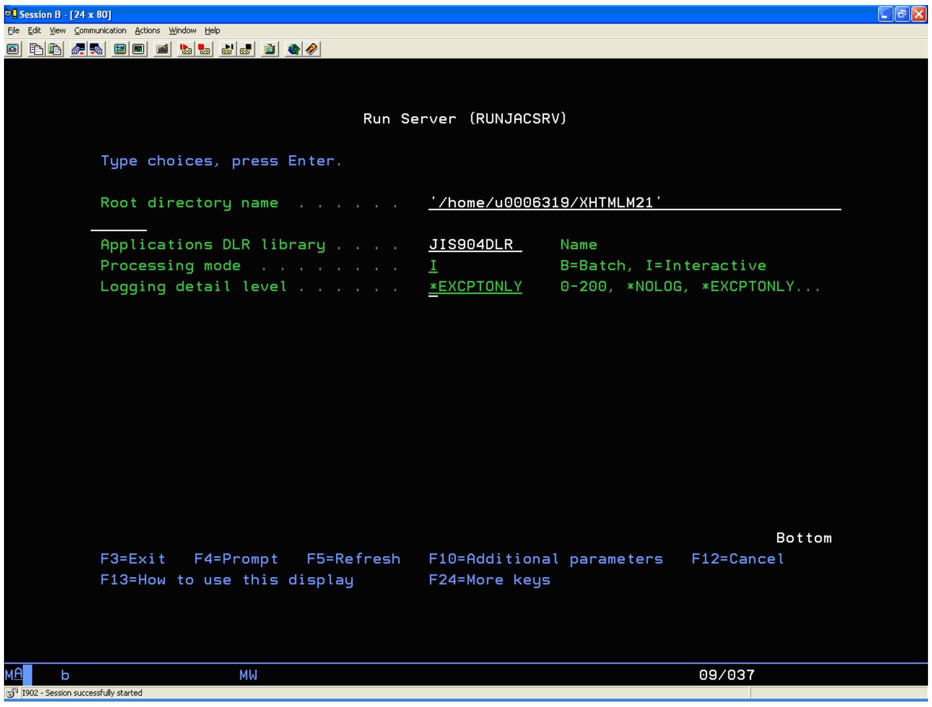
There are two ways you can run the JIS Server.

- By typing the `RUNJACSRV` command. This runs the Server and allows you to define two parameters to determine how the Server runs.
- By typing a customizable command. This is used for defining several parameters by which the JIS Server runs, and the class paths it reads.

Activating the JIS Server Using the RUNJACSRV Command

To activate the JIS Server using RUNJACSRV:

1 In the command line type the following command and press **Enter**:
`RUNJACSRV`
The following screen appears:



```
                              Run Server (RUNJACSRV)

 Type choices, press Enter.

 Root directory name . . . . . .   '/home/u0006319/XHTMLM21'
 ______
 Applications DLR library . . . .   JIS904DLR     Name
 Processing mode  . . . . . . . .   I             B=Batch, I=Interactive
 Logging detail level . . . . . .   *EXCPTONLY    0-200, *NOLOG, *EXCPTONLY...

                                                                         Bottom
 F3=Exit   F4=Prompt   F5=Refresh   F10=Additional parameters   F12=Cancel
 F13=How to use this display        F24=More keys
```

Figure 28. Run JIS Server (RUNJACSRV) screen

2 Fill in the fields according to the following instructions:

Field	Instructions
Root Directory Name	Enter the directory name under which the JIS Server is installed
Application DLR library	Put the name of the `DLR` library that you specified when you ran the `JSINSTALL` command. Define parameters that determine how the JIS Server runs. Note that this is optional.
Mode	Choose between **Batch** and **Interactive**. **Batch** - The Server is activated and works in the background **Interactive** - Activity taking place on the Server is displayed
Logging detail level	“Debug” level. See “The JIS Server Command Line Parameters” on page 103.

Depending on the capacity of the iSeries, it takes the Server up to a minute after invocation time before it is ready to serve clients.

The JIS Runtime on Solaris

In this section, we describe how to install and operate the webMethods JIS runtime on a Solaris machine. In doing so we have made some basic assumptions, which include:

- The Solaris machine is an integral part of your company’s network.
- If you use Samba as a means to install the runtime installation on a Solaris machine, it is assumed that your Systems Administrator knows to configure Samba so that a PC drive can be mapped to a directory on the Solaris machine.

Note: Commands given in UNIX are case sensitive. Take care to correctly enter directory paths and commands.

The section is divided as follows:

- Preparing the Solaris directory structure
- Creating a runtime installation
- Installing the runtime to the Solaris machine
- Activating the JIS Server on Solaris

Preparing the Solaris Directory Structure

You need to perform some operations on the Solaris machine to prepare it to accept and run the runtime installation. This initial preparation must be performed irrespective of whichever means you use to transfer and install the runtime installation onto the Solaris machine.

The operations include:

- Creating a User ID for the JIS Application
- Creating a working directory under the user directory

Creating a User ID for the JIS Application

Have the System Administrator create a User ID.

Creating a Working Directory Under the User Directory

The working directory serves to house the runtime deployment.

To create a working directory:

- From within the <user name> directory create a working directory:
  `/export/home/<user name>/<Working Directory>`

Creating a Runtime Installation

To create a runtime installation:

1. In ACE, from the **Utility** menu choose **Create Runtime Installation**.
   This opens the **Create Runtime Installation** wizard.
2. In the **Create Runtime Installation** wizard choose the following settings:
   - **Automatic or manual packaging**. It is recommended that you use the WISE packaging system. If you choose to use WISE, then the **Create Runtime Installation** wizard produces an information file that WISE can read. If you do not choose WISE, then the **Create Runtime Installation**

wizard produces a text file listing the directory structure and files that make up a working runtime on an end-user system.

- If you are using WISE then you can choose a bitmap to be displayed during the installation on the end user's system.
- For **Runtime Type**, choose XHTML.
- For the *JIS Server Platform(s)* option, choose **Solaris SPARC**.

3 Continue the wizard's steps to the end.

At the end of the process ACE creates the following files:

- **setupjav.exe**—an executable for the Wise Installation System software. This program installs the JIS Server for your runtime Application.
- **setup.txt**—if you choose to install the runtime without using the Wise installation program, this text file lists the files that should be included in the runtime Application. This text file also includes an indication of the precise place in the Application runtime directory in which each file should appear.

These files are placed under the ACE root directory, under the directory:
`<InstallDir>\appls\<ApplName>\install\javasrvr`

The entire runtime environment is contained in the `setupJav.exe` file. To distribute the webMethods JIS runtime, copy the `setupJav.exe` file to a distributable media

Pre-Installation Checklist for the Solaris Platform

Before deploying the webMethods JIS runtime on the Solaris machine, check the following:

- The runtime was generated for the Solaris SPARC platform.
- The Runtime Installation was created for the Solaris SPARC platform.
- Check the version of the operating system and JDK that are installed on the Solaris machine.
- Check the amount of physical memory available on the Solaris. Each application session requires approximately 1.5 MB.
- See that the Domain Name Server is properly configured.
- Verify that you have UNIX permission for transferring the runtime files and for executing the Server.
- See that you have a file sharing utility like Samba or NFS installed and configured on the Solaris; alternatively, check that you can transfer files to the Solaris via FTP.
- Check that JIS default ports 1100, 1101, and 2100 are not in use by another process on the Solaris and are not blocked by a firewall.

Installing Your Runtime on a Solaris Machine

There are several types of utilities you can use to transfer and install the runtime from the PC to the Solaris machine. These include:

- Samba
- FTP
- Other utilities

Installing the Runtime Environment Using Samba

Note: If for whatever reason you are not able to use a file sharing utility to map a network drive to your Solaris, skip ahead to "Installing the Runtime Environment Using FTP" on page 98.

Use the Samba utility to view and manipulate the Solaris machine's directory and file structure from the PC. In this way you can transfer the runtime installation file directly from the PC to the Solaris machine, and then deploy the runtime.

To use Samba in this way:

1 Designate a directory on the Solaris machine to view from the PC.
2 Map a drive on the PC to the Solaris machine.
3 Install the runtime environment on the Solaris machine.
4 Manipulate deployed runtime files.

Mapping the PC to Solaris

There are two ways to map a Network Drive to the Solaris Machine. You can map the drive using Network Neighborhood, or with Windows Explorer.

To map a PC drive to a Solaris machine using Network Neighborhood:

1 From the Network Neighborhood, find the name of the Solaris machine and double-click on it.
2 Locate the "Shared directory" on Solaris to which you wish to map the local drive. For example: `\\<host name>\home\jis`. Click on it with the right mouse button.
3 From the shortcut menu, select **Map Network Drive**.
4 In the **Map Network Drive** dialog box, choose a local drive.

To map a PC drive to the Solaris machine using Windows Explorer:

1 From the **Tools** menu in Windows Explorer, select **Map Network Drive**. The **Map Network Drive** dialog box appears.
2 Windows offers you the first available network drive. This is fine, unless for some reason you want to assign a specific drive—perhaps you want to use the “S” drive, for “Solaris”.
3 Chose the appropriate path to the “Shared directory” which has been assigned to you on the Solaris. If the desired path does not appear in the dropdown list you can type it manually into the Path field.
  `\\12.34.56.78\jis` **or, with a DNS name:** `\\OURSOLARIS\jis`
4 Click **OK**.

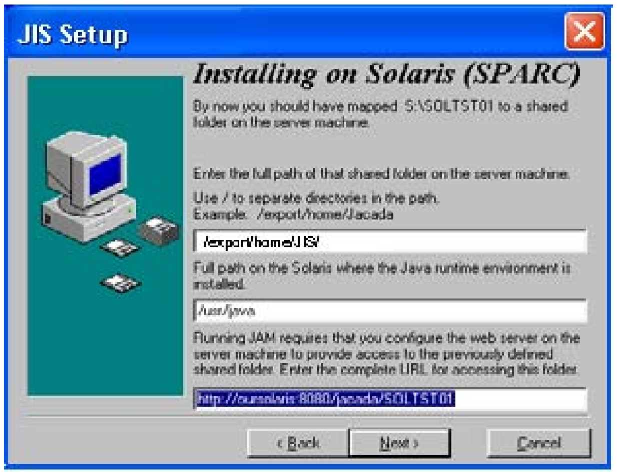
Note: You should record the Solaris directory path. You will need to enter it in the **JIS Server Installation** wizard.

You should now be able to view the contents of the <working directory> on the Solaris machine from the PC.

Transferring the Runtime Environment to the Solaris Machine

To transfer the runtime environment to the Solaris machine:

1 From the PC, run the webMethods JIS for XHTML runtime executable by activating the `setupJav.exe` file.
  This invokes the **JIS Server Installation** wizard.
2 In the **Select Destination Directory** dialog box, enter the directory under which the JIS Server files will be installed.
  Enter a directory according to the following criteria:
  - If you are using Samba, enter the runtime directory as defined when mapping your PC drive to the Solaris machine.
  - If you are using FTP, enter the name of a temporary directory.
  - When working with Solaris it is important to remember that UNIX is case sensitive. When creating an Application, ACE forces its name to uppercase. Therefore, where the Application name appears in a pathname in any of the runtime classes, it is also in uppercase, and the name of the directory where the Application resides must also be in uppercase.
3 The **Installing on Solaris (SPARC)** dialog box is displayed:



Installing on Solaris (SPARC)

By now you should have mapped S:\SOLTST01 to a shared folder on the server machine.

Enter the full path of that shared folder on the server machine.

Use / to separate directories in the path.
Example: /export/home/Jacada

/export/home/JIS/

Full path on the Solaris where the Java runtime environment is installed.

/usr/java

Running JAM requires that you configure the web server on the server machine to provide access to the previously defined shared folder. Enter the complete URL for accessing this folder.

http://oursolaris:8080/jacada/SOLTST01

Figure 29. RT transfer to Solaris

4 In the **Installing on Solaris (SPARC)** dialog box, set the following options:

Option	Description
The full path on the Solaris to the directory where the application is installed	If the directory for your work on the Solaris is: `/export/home/jis` and you have designated the sub-directory `SOLTST01` to contain the runtime environment, then the full Solaris directory for your runtime environment will be: `/export/home/jis/SOLTST01`
Full path on the Solaris to the Java runtime environment	The default is: `/usr/java,` the installation will automatically append `/bin/java` to this path.
URL for accessing the application	A URL including the Solaris machine's address followed by the directory to which you are transferring the runtime environment, in the following format: `http://<your Solaris's address>:8080/<shared folder>/<appl directory>/` *Example:* `http://oursolaris:8080/jis/SOLTST01/`

The installation wizard automatically transfers and installs the runtime environment to the directory you have defined. In doing so the installation wizard installs both JIS Server and XHTML client for your runtime application.

Installing the Runtime Environment Using FTP

A simpler installation technique was introduced in JIS 9.0.3. Refer to the JIS 9.0.3 release notes for more information. This section is for those who are unable to use Samba or another file sharing utility. If you have successfully installed the Runtime Install using a file sharing utility, skip forward to "Activating the JIS Server on Solaris" on page 101.

Use FTP as a means to transfer the runtime installation from a drive on the PC to the Solaris machine in situations where you cannot use a utility such as Samba. Before transferring the runtime via FTP you must first install the runtime on one of your PC's local drives. You can then zip the runtime environment and FTP it to the Solaris machine.

All modifications to the Solaris machine's directory structure described in "Preparing the Solaris Directory Structure" on page 93, must be performed prior to transferring the compressed runtime environment.

To install the runtime environment on the Solaris machine:

1 Install the runtime environment on a temporary PC directory.
2 Compress and transfer the runtime environment to the Solaris machine.
3 Deploy the runtime environment into a pre-defined working directory.
4 Manipulate deployed runtime files.

Installing the Runtime Environment on a Temporary PC Directory

Run the installation file, `setupjav.exe`, to install the runtime environment on a directory on your PC. Running the installation file starts the **JIS Server Installation** wizard.

The wizard has the following steps:

- Registration Information
- File Transfer Method
- Select Destination Directory
- Installing on SPARC/Solaris

In the Registration Information step, enter:

- User Name
- Company Name

In the File Transfer Method step, enter:

- FTP

In the Select Destination Directory step, enter:

- The directory under which the JIS Server files will be installed on the PC. It may be useful to install under a directory structure that mirrors the structure that you established on the Solaris machine.
- The drive and directory on the PC that will act as a temporary location for deploying the runtime. Where the destination directory is `DEMO` and the working directory is `JIS`, enter: `<Drive>:\JIS\DEMO.`

In the Installing on SPARC/Solaris step, provide:

- Full directory path to destination directory as viewed on the Solaris machine. Where you have designated the directory `DEMO` as the destination directory, then the full Solaris directory for your runtime environment is:

  `/export/home/<UserName>/<WorkingDir>/<DestinationDir>`

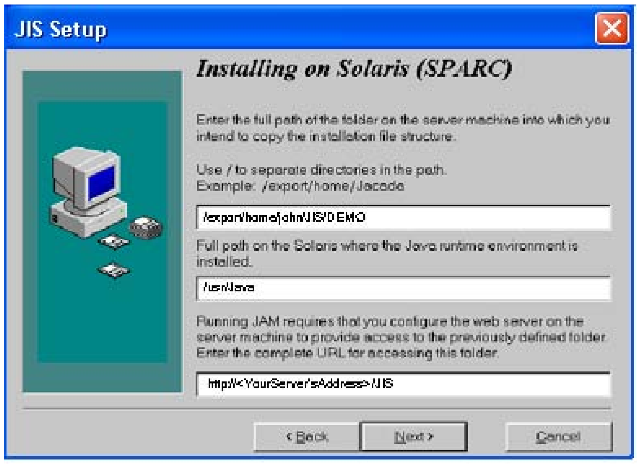
  Example: `/export/home/john/JIS/DEMO`
- Full path on Solaris where the Java utilities are installed. The default is:

  `/usr/java`
- A URL including the Solaris machine's address followed by the path to the Destination Directory:

  `http://Solaris/~<user name>/<Destination Directory>`

Note: All directory path entries refer to the installations destination on the Solaris machine and not the temporary directory being used on the PC.

If the administrator installed the JDK in a non-standard manner then you need to change this path to reflect the directory on the Solaris machine that houses the JDK.



JIS Setup

Installing on Solaris (SPARC)

Enter the full path of the folder on the server machine into which you intend to copy the installation file structure.

Use / to separate directories in the path.
Example: /export/home/Jacada

/export/home/john/JIS/DEMO

Full path on the Solaris where the Java runtime environment is installed.

/usr/Java

Running JAM requires that you configure the web server on the server machine to provide access to the previously defined folder. Enter the complete URL for accessing this folder.

http://<YourServer'sAddress>/JIS

Figure 30. RT transfer to Solaris on a temporary PC directory

The installation wizard automatically transfers and installs the runtime environment to the directory you have defined. In doing so the installation wizard installs both the JIS Server and the Client for your runtime application.

Compressing and Transferring the Runtime Environment to the Solaris Machine

To compress and transfer the runtime environment to the Solaris machine:

1. From the temporary directory established on the PC, select and compress the runtime environment into a ZIP or TAR file.
2. Open an FTP session with the Solaris machine.
3. FTP the compressed runtime environment to your target Solaris machine.

Deploying the Runtime Environment into a Pre-defined Working Directory

The next step is to deploy the runtime environment on the Solaris machine.

To deploy the runtime environment into a pre-defined working directory:

1. Using Telnet or any other means, open a session on the Solaris machine.
2. Navigate to the location where the compressed file is held.
3. UNZIP or "un-TAR" the runtime environment from the compressed file into the pre-defined working directory.

Activating the JIS Server on Solaris

The file system on the Solaris machine is case-sensitive. Therefore, before you run the JIS Server on Solaris, you should be aware of restrictions regarding case sensitivity.

Make sure that:

- References to files are written exactly like the files themselves.
- All INI files are written in lower case.
- Any directory with the same name as the Application is in uppercase, and the Application name itself is always uppercase.

To run the JIS Server from Solaris:

1. Open a session on the Solaris machine and enter your user name and password.
2. Change the directory to the 'bin' directory under which the runtime environment is installed. Type: `cd /<RT directory>/bin`

Note: <RT directory> must be replaced with the directory in which the JIS installation resides.

3 Make a backup copy of the `jacadasv` file by executing the following command: `cp jacadasv jacadasv_copy`

4 Execute the `dos2unix` command to change the end-of-line characters in the `jacadasv` file from MS DOS-style to UNIX-style:
`dos2unix jacadasv_copy jacadasv`

5 If this is the first time you are running the JIS Server, after having installed, you must grant the user permission to execute certain JIS Server files.

Type: `chmod u+x jacadasv solaris/*.so`

-OR- type: `chmod 755 jacadasv solaris/*.so`

JIS

6 Run the JIS Server by typing the following command: ./`jacadasv` from the directory under which the runtime environment is installed.

7 Start up a web browser and enter the following URL in the address bar:
`http://<your Solaris’s IP addr>:8080/<ApplName>-xhtml.html`

Note: You can have the JIS Server started and running in the background during the Solaris’s initialization process. To do so, add the executable command to the initialization path and add an ampersand (&) at the end of it—jacadasv&.

The Jacadasv Script

The `jacadasv` shell script contains parameters relating to the JIS Server executable file. These parameters can be changed.

The `jacadasv` script contains the following parameters:

```
set CST_DIR=/<runtime_directory>/
set JAVA_INTERPRETER=/usr/java/bin/jre
set JAVA_CLASSES=/usr/java/lib/rt.jar
if ( $?LD_LIBRARY_PATH ) then
   setenv LD_LIBRARY_PATH $CST_DIR/bin/solaris:
   $LD_LIBRARY_PATH
else
   setenv LD_LIBRARY_PATH $CST_DIR/bin/solaris
endif
limit descriptors unlimited
exec $JAVA_INTERPRETER -classpath $JAVA_CLASSES\: $CST_DIR/
classes:$CST_DIR/classes/cst/jacadasv.zip: $CST_DIR/utils/xml/parser/
xml.jar cst.server.module.ServerStarter -d0 $*
```

The JIS Server Command Line Parameters

You can manipulate the way the JIS Server runs by adding command lines to its script file.

Table 12 lists the parameters you can use for this purpose:

Table 12. JIS Server command line parameters (Sheet 1 of 2)

Parameter	Description
`-i <Initialization file path name>`	Default: `-i<User’s working directory>jacadasv.ini` *Example:* `-ic:\JacadaFiles\jacadasv_ex.ini`
`-d <Debug level>`	Default: `-d1` *Example:* `-d30` The debug level can be set to any integer from zero to 1000. The greater the integer, the greater the amount of information that is recorded in the logfile. A debug level of “70” produces an extremely detailed log file. A debug level of “1” is recommended when running the server in production mode. The debug facility is extremely useful for diagnosing problems that may occur during setup and testing of your JIS Server, but Software AG recommends that debug level larger than 10 will not be used on a regular basis during normal production operation. This is because the JIS Server generates many log entries for each action (every time “Enter” or an Fkey is pressed) of every user. Especially with the higher debug levels, a handful of users with moderate activity could result in an enormous log file in just a short time. After a certain point the logging of such a large number of entries may negatively impact system throughput.

Table 12. JIS Server command line parameters (Sheet 2 of 2)

Parameter	Description
`-l <Debug logging file directory>`	Default: `log to console` *Example*: write log to directory \temp on the c: drive `-lc:\temp` The logfile directory you specify must exist before you bring up the JIS Server. If the directory does not exist, the Server startup fails.
`-h`	The JIS Server console displays the syntax of the webMethods JIS startup command, and the list of command line options.
`-n`	Disables the user's option to insert such commands as '`check`' and '`quit`' in the JIS Server console. This flag is neccessary when running the server as a backround process.
`-c`	Runs the Server Configuration Checker in Offline Mode. The Checker analyzes the configuration of all the defined server machines, reports errors and warnings to a log, and closes without starting the server.

`-f <Debug Filters>`
`(if specifying multiple filters, separate them with a comma [","])`
	Debug Filters are tools to help you accomplish specific types of logging. See "Debug Filters" on page 166 for more information.

Chapter 5. Optimizing the JIS Server

The following topics are discussed:

- "JIS Server INI File Settings" on page 105
- "Scalability" on page 139
- "JIS Server Logging Support" on page 159
- "Analyzing Abnormal Runtime Termination" on page 168
- "Checking Server Configuration" on page 178
- "JIS Administrator" on page 184
- "Running the JIS Server as a Windows Service" on page 218
- "Printing Emulation" on page 222

JIS Server INI File Settings

This section provides you with a list of the parameters that can be set in the JIS Server initialization file (JIS Server INI file). The actual file name of the JIS Server INI file is `jacadasv.ini`, and it is located in `<JIS Root Directory>\JacadaFiles\classes`.

Many of the parameters in the JIS Server INI file deal with aspects of connectivity between the JIS Server and the XHTML client Application. For many of the parameters, the default values are satisfactory and need never be modified. For other parameters, you should supply custom values reflecting your environment.

Location of INI Settings: A Recommendation

Beginning with webMethods JIS version 9.0, Software AG recommends that INI settings that have until now been defined in the XHTML section of the `jacadasv.ini` file, be moved to the XHTML section of the runtime INI file (`<APPLNAME>.ini`). This is to ease the task of deploying your applications to a J2EE application server, if you should choose this path in the future.

In a J2EE deployment, the `jacadasv.ini` file contains only debug-related settings. For any of the XHTML section's INI settings to be effective under J2EE, they must be located in the runtime INI file (`<APPLNAME>.ini`).

Under the webMethods JIS proprietary server, the only difference between an INI setting in the `jacadasv.ini` file and one in the runtime INI file is the setting's scope. A setting in the `jacadasv.ini` applies to all applications running under the JIS Server; a setting in the runtime INI file applies only to the application associated with that runtime INI file. If an XHTML setting is defined in both the `jacadasv.ini` file and the runtime INI file, the runtime INI value takes precedence.

The remainder of this chapter is written from the perspective of a deployment to the JIS standalone Server; that is, it assumes the settings are to be placed in the jacadasv.ini file. However, please keep the above recommendation in mind.

Other Factors Affecting Performance

Several factors can affect the performance of the JIS Server in addition to the Server's settings and the users' activity profiles.

- The Server platform's CPU and memory configuration has the most profound impact on capacity and performance.
- Network capacity and congestion directly affect performance.
- Optimization of load balancing configuration parameters can dramatically affect performance.
- The performance of the JVM also influences overall performance.

The INI Settings

The JIS Server INI file is called `jacadasv.ini`. It resides under the Classes directory, in the directory in which you have installed your runtime Application.

For example: `c:\JacadaRuntime\classes\jacadasv.ini`

Note: For changes to take effect you should restart the JIS Server.

The INI file is divided into sections. Each section begins with a header line that consists of the name in the section in square brackets, like so: `[SectionName]`. This section of the book is organized into sections like those in the INI file itself.

[GeneralParameters]

Table 13. Jacadasv.ini: [General Parameters] section (Sheet 1 of 9)

Parameter	Description
`BindIPAddressFor-GUIClient`	Defines the IP address(es) on which the JIS Server listens. The following values are available: **_All_** - listens to all the IP addresses available on a specific machine. This is the default value. **_AsListed_** - listens to the IP address(es) listed in the `[ServerMachines]` section of the `jacadasv.ini` file. **ExplicitIPAddress** - listens to an explicit IP address available on a specific machine.
`CheckServerConfiguration`	On startup, checks the configuration of the currently running server. To disable, set to 0. Default value is 1.
`HTTPClient`	Defines whether or not to enable HTTP Client support. 1 = enable HTTP Client support 0 = disable HTTP Client support Default value: 1 **Note:** In order to enable the XHTML client, this value must be set to 1.
`InitialTransactionsTo Ignore`	This parameter is used in calculating the average duration of transactions, for logging purposes. The initial transactions' duration are longer due to Java class loading, so you have the option of not counting them when calculating the average. Defines the number of initial transactions that are ignored. The default value is 1.

Table 13. Jacadasv.ini: [General Parameters] section (Sheet 2 of 9)

Parameter	Description
`IniVersion`	Optional parameter. The developer can use this parameter to specify a character string that appears in the debug log to identify the ini file being used.
`KeepAliveTimeout`	Timeout for sub-processes to send a keep-alive notification to the main process. If the main process does not receive a keep-alive notification before the timeout expires, the reference to the sub-process is removed. The value is calculated as multiples of the `KeepAliveTimerTick` parameter. Default value: 10 (x `KeepAliveTimerTick`).
`KeepAliveTimerTick`	Interval at which sub-processes send a keep-alive notification to the main process. Default value: 60 seconds.
`MaxMachineApplications`	Maximum number of applications to be run concurrently on the server machine. Default value is no limit.
`MaxProcessApplications`	Maximum number of applications to be run concurrently in the server process. Default value is no limit.
`MaxProcesses`	Maximum number of processes to be run concurrently on the server machine. Default value: 1
`PortScanRetries`	Defines the number of times the process looks for a free port within the port range before giving up. Default value: 2
`ProcessRespawnEnabled`	In case of system failure, defines whether or not to destroy and re-spawn corrupted processes. Valid values: 0 (no), 1 (yes). Default value: 0.

Table 13. Jacadasv.ini: [General Parameters] section (Sheet 3 of 9)

Parameter	Description
`RegistryPortRange`	Range of port allocations for the RMI Registry (minimum of one port is needed). Format: <low port> - <high port> *Example*: 1900-1901 Default port number is 2100.
`RegistrySpawnTimeout`	The time in milliseconds the server waits for the NodeRegistry process to start. Default is 45000 milliseconds. Do not use a comma when specifying the value.
`ReportsToMachine`	Alias of the root machine. **Note:** required only in multi-machine scenarios, for machines that are not running the root server process. The machine running the root process must remain undefined.
`ReverseDNS`	Enables the DNS name lookup in a DNS server. When set to 1, the client's host name is written to the SessionLog and XMLLog file. Valid values: `1 | 0`. Default setting 0, writes the numeric IP address to the log.
`RMISocketTimeout`	Defines the time in seconds that an RMI client waits for a server to respond. When a remote method is called on a remote object it times out after the defined time, throwing a 'RemoteException'. Default value: 20 seconds

Table 13. Jacadasv.ini: [General Parameters] section (Sheet 4 of 9)

Parameter	Description
`RtDebugFileMaxSize`	Specifies the maximum size of the server log file in bytes, if desired. When the logfile reaches this size, wraparound occurs. *Example:* `RtDebugFileMaxSize=1000000` Limits the webMethods JIS log file to 1 Mb in size. The maximum size of the server log file can also be specified on the JIS Server command line (See "The JIS Server Command Line Parameters" on page 103.) If the parameter is specified in both places, the value specified on the command line takes precedence.
`RtDebugFilters`	Specify any debug filters that you want set on automatically at Server startup. *Example*: `RtDebugFilters=SCREEN_IMG_NL` If you desire to use more than one debug filter, separate the filter names with commas. Debug Filters are tools to help you accomplish specific types of logging. See "Debug Filters" on page 166 for more information. Debug filters can also be specified on the JIS Server command line. (See "The JIS Server Command Line Parameters" on page 103.) If debug filters are specified in both places, the values specified on the command line take precedence. Debug filters can also be turned on and off while the JIS Server is running. See "Debug Filters" on page 166 for more information.

Table 13. Jacadasv.ini: [General Parameters] section (Sheet 5 of 9)

Parameter	Description
`RtDebugLevel`	The debug level can be set to any integer from zero to 1000. The greater the integer, the greater the amount of information that is recorded in the logfile. A debug level of "70" produces an extremely detailed log file. The debug level can also be specified on the JIS Server command line (See "The JIS Server Command Line Parameters" on page 103.) If the debug level is specified in both places, the value specified on the command line takes precedence. The debug facility is extremely useful for diagnosing problems that may occur during setup and testing of your JIS Server, but Software AG recommends that debug mode not be used on a regular basis during normal production operation. This is because the JIS Server generates many log entries for each action (every time **Enter** or an Fkey is pressed) of every user. Especially with the higher debug levels, a handful of users with moderate activity could result in an enormous log file in just a short time. After a certain point the logging of such a large number of entries may negatively impact system throughput.

Table 13. Jacadasv.ini: [General Parameters] section (Sheet 6 of 9)

Parameter	Description
`RtLogsDir`	Use this parameter to specify the location of the log files. For example: `RtLogsDir=$RootDir\classes\logs` Alternatively, you can specify the log file location on the command line. (See "The JIS Server Command Line Parameters" on page 103.) The debug level must be higher than zero for a Server Log File to be created. The log directory specified (either via the `RtLogsDir` parameter or on the command line) must already exist in order for logging to take place. The debug file itself is called `Debug_1.log`. If there is more than one server session, a numeric suffix is automatically affixed so that the file name becomes `Debug_1.0.log`, for example.
`RtRootDir`	The runtime root directory. `C:\ACE\JacadaFiles` **Note:** The string `"$RootDir"` in settings that include it in their path are replaced in runtime with the root directory defined in this setting.

Table 13. Jacadasv.ini: [General Parameters] section (Sheet 7 of 9)

Parameter	Description
`ServerPortRange`	Range of port allocations for the initial communication port number. A minimum range of at least one port is required. **Note:** If no `ServerPortRange2` parameter is specified: - This range also applies to the secondary server port. For this to be effective, a minimum range of at least two ports is required. - When the `ServerPortRange` parameter is set to a range of only two ports, the lower value of the range is reserved for port1 and the higher value is reserved for port2. Format: <low port> - <high port> *Example:* 1900-1901 Default value: 1100-1100. **Note**:this setting is only used by the Java Client
`ServerPortRange2`	Optional range for the secondary communication port number. Format: <low port> - <high port> Example: 1900-1901 Default is the port range defined in `ServerPortRange`. **Note**:this setting is only used by the Java Client

Table 13. Jacadasv.ini: [General Parameters] section (Sheet 8 of 9)

Parameter	Description

`SocketImplFactory`	The fully qualified classname of a custom implementation of the socket factory class on the server side.
	
	`appls.applname.server.user.`
	`MySocketFactoryImp`
	
	Note: When undefined, uses regular Java sockets.

`SoftLimitMarginPercent`	An integer that defines a secondary "soft" limit below the maximum sessions limit. When the soft limit is reached, the main node stops directing new clients to this node until all nodes have reached this value. The node, however, does not reject clients that contact it directly. The value is defined as a percentage of `MaxProcessSessions`. Do not type the percent sign. Default value is 10.
`SpawnInterval`	Time, in milliseconds, that the main server process waits before spawning the next process. The default value is zero, which means that the main process does not wait before spawning other processes.
`StartScanAtRandomPort`	To avoid a situation in which all processes vie for the same free port, this setting defines a random starting point within the range of allocated ports for each process. If the port range is 2000-3000 and the random starting point is 2500, then the process starts looking for the free port at port 2500-3000, and if not found, from port 2000-2499.

Table 13. Jacadasv.ini: [General Parameters] section (Sheet 9 of 9)

Parameter	Description
`StdoutEncoding`	Specifies the Encoding setting the Server processes use when reading the output of spawned processes. This is dependent on the iSeries operating system. OS V4R2 requires the following setting: `Stdout=Cp037`
`SystemConnection Timeout`	On startup, the interval in seconds for which the main node tries to establish a connection with a root node on another machine. Default value is 120.
`WaitForSpawned`	Maximum waiting time in seconds for a server process to start and initialize. After this interval the spawned process is considered to have failed. Default value is 60.

[HTTP]

Table 14. Jacadasv.ini: [HTTP] section (Sheet 1 of 4)

Parameter	Description
`ExternalErrorPages Dir`	Specifies the directory where your custom-written HTML error pages reside. Used in conjunction with the `UseExternalErrorPages` parameter. Code this parameter as follows, using the folder names shown, substituting your root directory for the first node: `<InstallDir>\JacadaFiles\classes\HttpErrors`

Table 14. Jacadasv.ini: [HTTP] section (Sheet 2 of 4)

Parameter	Description
`HTTPPortRange`	Specifies the port range of the JIS Server's HTTP connection. Format: `<low port>-<high port>` *Example*: 8080-8084 Default: 8080-8080
`HTTPSPortRange`	Defines the ports which listen for HTTP**S** requests. Use different ports than those used for HTTPPortRange. Format: `<low port>-<high port>` *Example:* `8085-8089` Default: `8443-8443`
`JettyConfiguration File`	Setting indicates that jetty loads the `proxyHttp.xml` configuration file which tells jetty to run the redirection proxy. See "Redirection Proxy" on page 153 Valid value: `proxyHttp.xml` Default value: `http.xml`
`MaxSession InactivityTimeout`	Maximum period of inactivity, in minutes, after which the session is ended by the Server. Default is 60 minutes. In regards to this parameter, the "keep alive" messages automatically sent by the client are **not** considered a user action, and do **not** cause the MaxSessionInactivityTimeout timer to be reset. This parameter is defined in the HTTP section of the `jacadasv.ini` file. See also "Keep Alive Implementation for the XHTML Client" on page 414.

Table 14. Jacadasv.ini: [HTTP] section (Sheet 3 of 4)

Parameter	Description
`NeedClient Authentication`	Defines whether the server requires client authentication for an HTTPS connection. This means that the client must present a certificate in order to prove its identity. Valid values: 1 (yes), 0 (no). Default: 0 This new setting is effective only if `SupportHTTPS=1`. webMethods JIS does not provide the trusted certificates. To use this feature you must insert the trusted certificate in the JettyKeyStore. Contact Software AG Support for information on how to do this.
`ProxyConfiguration File`	This parameter is relevant if you are using the redirection proxy. See “Redirection Proxy” on page 153. Valid value: `<InstallDir>\JacadaFiles\classes\proxyconfiguration.xml` Default value: null
`ResourceBase`	Specifies the base directory from which the location of image files is defined. The value of the `[XHTML]` section’s `ImagesLocation=` parameter relates to this directory. Usually, this is the runtime root directory. `C:/ACE/JacadaFiles`
`ResponseTimeout`	Defines the time in milliseconds that HTTP support waits for the method to complete writing the response, before aborting the action. Default is 100000 (100 sec.).

Table 14. Jacadasv.ini: [HTTP] section (Sheet 4 of 4)

Parameter	Description
`SessionIdleTimeout`	Specifies, in seconds, how long an HTTP session remains open if the end-user makes no submission to the JIS Server. In regards to this parameter, the “keep alive” messages automatically sent by the client are considered user actions, and cause the SessionIdleTimeout timer to be reset. The default inactivity time-out of XML sessions is configurable via the `SessionIdleTimeout` property in the `ServerConfiguration.xml` file for the JIS Server, and via the `session-timeout` parameter in the `web.xml` file when using a J2EE application server. The installation default for both of these files is 180 seconds (3 minutes). Under the proprietary JIS Server, the Server looks for the `SessionIdleTimeout` value in the `ServerConfiguration.xml` file first. If it does not find it there, it then looks in the [HTTP] section of the `jacadasv.ini` file. If the value is not defined there, then the default value of 3600 seconds is used. See also “Keep Alive Implementation for the XHTML Client” on page 414.
`SupportHTTPS`	Provides support for HTTPS, when set to 1. Valid values: 1 (yes), 0 (no). Default: 0 The HTTPClient parameter in the `[GeneralParameters]` section must be set in order to use HTTPS.
`UseExternalErrorPages`	Valid values: 1 (yes), 0 (no) This parameter affects the Server’s response to HTTP errors. When this parameter is set to 1, if an HTTP error occurs, the Server looks in the directory specified by the `ExternalErrorPagesDir` parameter for a user-written HTML page named `<error_code>.html` and displays it instead of the standard error page.

[XHTML]

Table 15. Jacadasv.ini: [XHTML] section (Sheet 1 of 5)

Parameter	Description
`DefaultButton`	Use to enable or disable default button functionality. 0 = Disables default button functionality 1 = Enables default button functionality Default value is 1. **See the note at the end of this section.**
`DILPosition`	Specifies whether the DIL messages appear on the top or the bottom of the web page. Valid values: `Top | Bottom`. Default is `Bottom`.
`DisplayHostImage WhenOutOfSync`	Specifies whether the **Out of Sync** error page includes the screen currently displayed on the host. 0 = Do not display host screen. 1 = Display host screen.

Table 15. Jacadasv.ini: [XHTML] section (Sheet 2 of 5)

Parameter	Description
`DocTypeDeclaration`	Sometimes the `doctype` and `xml` declarations in our XHTML forms cause problems; for example, they can cause the browser to display the html source instead of a GUI object. The `xml` declaration looks like this: <?xml version="1.0" encoding="UTF-8"?> The `doctype` declaration looks like this: <!DOCTYPE html SYSTEM "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional"> This INI setting lets you control whether or not `doctype` and `xml` declarations appear in the XHTML form. Valid values are: 1 (add the declarations) and 0 (do not add them). Default value is 1.
`DoHTMLMerge`	Specifies whether or not to enable HTML extensions. 0 = disable HTML extensions 1 = enable HTML extensions Default is 0.
`FkeySupport`	Indicates whether the client should support Fkey usage or not. 1 = yes, 0 = no. Default value is 1. This setting is used in the XHTML section of the `<APPLNAME>.ini` file. It makes Fkeys PF1 through PF24 available to subapplication methods, but not other accelerator keys such as Ctrl, Alt, Page Up, and so on.

Table 15. Jacadasv.ini: [XHTML] section (Sheet 3 of 5)

Parameter	Description
`ImagesLocation`	Specifies the location of application image files as they are accessed through the web server For handling images through the internal Jetty HTTP Server, specify the relative path to the image location. Enter: `/classes/appls` Default: `<Server Address>/classes/appls`.
`JavascriptLocation`	Specifies the directory where the JIS JavaScripts reside. This parameter works slightly differently depending on the setting of the `RunAsWebApp` parameter (also part of the XHTML section). `The RunAsWebApp` parameter indicates whether the application is to run on the webMethods JIS proprietary server or on a J2EE application server. **For the JIS Server:** The default is `<InstallDir>\classes\js` If you specify a value, `\classes\js` is automatically concatenated to the value. **For a J2EE application server:** The default is `<InstallDir>\<applname>\js` If you specify a value, `\<applname>\js` is automatically concatenated to the value. *Example*: `JavascriptLocation=c:\appserver\domains`

Table 15. Jacadasv.ini: [XHTML] section (Sheet 4 of 5)

Parameter	Description
`KeepAliveInterval-InSeconds`	Time interval in seconds at which the XHTML client sends “keep alive” messages to the server. Default value is 60. This parameter can be defined either in `jacadasv.ini` or in the runtime *.ini file (`<applname>.ini`). See also “Keep Alive Implementation for the XHTML Client” on page 414.
`OutOfSync`	The behavior of the application when it encounters an “out-of-sync” condition can be customized. An out-of-sync condition is an error condition in which the screen being seen by the end-user does not represent the current host screen. Valid values: • **Page** – Sends an “out-of-sync” message page to the user.· • **StatusLine** – Sends the next page to the user, and add a message to the status line of the browser.· • **Silent** – Sends the next page to the user without letting him know what happened. Example: OutOfSync=Silent The default value is **Page** so that the behavior of existing applications does not change.The default for new applications is **StatusLine**.

Table 15. Jacadasv.ini: [XHTML] section (Sheet 5 of 5)

Parameter	Description
`PopupSupport`	Valid values: 0 | 1 The default value for existing applications (if the entry does not exist) is 0. For new applications, the value of 1 is automatically defined in the newly created runtime INI file. When enabled, in “New Subapplication Wizard” you can define a subapplication as a pop-up window. When a host screen is identified as a popup window, a new browser window is open on top of the already existing window. The existing page is not refreshed, and all elements of the existing page are disabled.When a popup window is closed, the window beneath it is enabled. When the main window is closed, all open popup windows are closed as well.
`RMBSupport`	Valid values: 0 | 1 This setting indicates whether the Right Mouse Button popup menu feature is to be supported. 0 = RMB popup menus not supported 1 = RMB popup menus supported The default value is 1.
`RuntimeDirectory`	Specifies the location of runtime-generated Subapplication XHTML files. Default value: `<InstallDir>\classes\appls`
`ShowTableUpDown Buttons`	Specifies whether the Up and Down buttons are shown on tables. Valid values: 0 (no), 1 (yes) Default value: 1

Note: A button is a "Default" button if developer defines it as such in the button style tab. In a Subapplication, if the focus is on a text field, a button, a check box, or a Radio control, and the user presses the **Enter** key, the XHTML client submits the form as if the default button was clicked. If the Subapplication contains more than one button which is defined as a default button, the first one is the effective one.

[Applications]

Table 16. Jacadasv.ini: [Applications] section

Parameter	Description

`Application_Name1`
 	
`Application_Name2`
	A list of Applications that are installed on the JIS Server. For each Application listed in this section, there should be a separate section created with the same name as the Application.

[<ApplName>]

One of these sections is created for each Application listed in the `[Applications]` section.

Table 17. Jacadasv.ini: [<ApplName>] section

Parameter	Description
`WorkingDirectory`	The full path to the application's resources. *Example:* `$RootDir\classes\appls\<ApplName>\server\resources\`
`IniDir`	The full path to the application initialization files. *Example:* `$RootDir\appls\<ApplName>\rt32`

[VMCommandLine]

Table 18. Jacadasv.ini: [VMCommandLine] section

Parameter	Description
`Classpath`	The classpath used by the Java VM for spawned processes. **Note**: The existing Server '-classpath' command line option (or its equivalent in accordance with the VM implementation in use) has to be prepended to the parameter value.
`JavaMemory`	Subset of the Java VM specific command line options used for memory settings. *Example*: `-ms10m -mx50m` **Note:** If not set, the Java VM uses its own defaults.
`JavaOptions`	Java VM specific command line options. **Note**: If not set, the Java VM uses its own defaults. *Example*: `-Djava.security.policy=$RootDir\classes\jacadasv.policy`
`JavaVM`	File path of the Java VM used for spawning processes. *Example*: `$RootDir\utils\jre16\bin\java`

[ServerMachines]

Table 19. Jacadasv.ini: [ServerMachines] section

Parameter	Description
`(M1, M2,)`	Associates the machine name/IP address/local host with a user-defined alias. *Example:* `191.96.15.2=M1`

[Sessions]

Table 20. Jacadasv.ini: [Sessions] section

Parameter	Description
`MaxProcessSessions`	Maximum sessions to be allocated in a server process.
`MaxMachineSessions`	Maximum sessions to be allocated in a server machine.
`StartUpSessions Percent`	Specifies the portion of `'MaxMachineSessions'` that must be provided by the server on start-up. Server processes are spawned in order to provide the required space for session allocation. Note that when the value=100, dynamic process spawning is disabled. Default is 0. **Note:** This parameter is similar to `SpareSessionsPercent` with the exception that this parameter has effect only on startup. The actual number of processes is limited by the `MaxProcesses` parameter.
`SpareSessions Percent`	Specifies the portion of `MaxMachineSessions` that must be provided during runtime. Server processes are spawned in order to provide the required space for session allocation. Default is 0. **Note:** The sessions designated by this parameter are allocated below the margin defined in the `SoftLimitMarginPercent` parameter.

[SessionTimeouts]

Table 21. Jacadasv.ini: [SessionTimeouts] section(Sheet 1 of 2)

Parameter	Description
`MsgboxTimeout`	Time in seconds that the Server waits for the Client to respond to a message box before terminating the session. Default is 36000.

Table 21. Jacadasv.ini: [SessionTimeouts] section(Sheet 2 of 2)

Parameter	Description
`PanelTimeout`	Time in seconds that the Server waits for the Client to select a panel (when working in File mode), before terminating the session. Default is 36000.
`GetTextFromUser Timeout`	Time in seconds that the Server waits for the Client to reply to a `GetTextFromUser` prompt before terminating the session. Default is 36000.
`KeepAlive`	Time in seconds during which the Server checks the connection with the Client. If the Client does not respond within `RecvTimeout`, the session terminates. Default is 240. **Note**: This setting is only used by the Java Client.
`RecvTimeout`	Time in seconds that the Server waits for a response from the Client before terminating the session. This applies to server messages that require a reply from the client, such as `KeepAlive`. Default is 200. **Note**: This setting is only used by the Java Client.

[ScoreWeights]

These parameters are used in load balancing and scalability calculations.

Table 22. Jacadasv.ini: [ScoreWeights] section (Sheet 1 of 2)

Parameter	Description
`MachineApplications`	The penalty weight of the number of running applications on the server machine. Specify as an integer, without percent sign. Default value is 0 percent of the total score weight.
`MachineSessions`	The penalty weight of the number of running sessions on the server machine. Specify as an integer, without percent sign. Default is 45 percent of the total score weight.

Table 22. Jacadasv.ini: [ScoreWeights] section (Sheet 2 of 2)

Parameter	Description
`MultiMachines`	The penalty difference that overrides the 'Preferred Machine' feature. If the penalty difference of two processes on two machines is greater than the specified value, then the session allocation occurs in the process with the lower penalty score. Specify as an integer, without percent sign. Default is 75 percent.
`ProcessApplications`	The penalty weight of running a new application in the server process. Specify as an integer, without percent sign. Default value is 10 percent.
`ProcessSessions`	The penalty weight of the number of running sessions on the server process. Specify as an integer, without percent sign. Default is 45 percent.

[ProcessCheck]

Table 23. Jacadasv.ini: [ProcessCheck] section (Sheet 1 of 2)

Parameter	Description
`CheckEnable`	When set to 1, enables the checking function of the integrator process. This ensures the server system integrity on each machine. Default is 0.
`CheckInterval`	Interval in seconds between server integrity checks. Default is 60.
`CheckProcessTimeout`	When integrity checking is enabled, the server makes sure that the processes are checked regularly. Otherwise, when this timeout expires, the server process starts the integrator recovery procedure. The timeout is calculated as multiples of `CheckInterval`. Default is 3.

Table 23. Jacadasv.ini: [ProcessCheck] section (Sheet 2 of 2)

Parameter	Description
`CheckTimeout`	Time in seconds that the integrator waits for a server process to respond to a check request before declaring the process as “corrupted”. Default is 30 seconds.
`IgnoreOpenSessions`	When set to 1, a server process declared “corrupted” by the integrator is shut down even if there are currently sessions running on the process. By default, a server process waits for the sessions to close down first.
`KillFaultyProcess`	This setting specifies whether or not a process should be killed if it has become corrupted (stalled); that is, if it has not responded within the period of time specified by `CheckTimeout`. Valid values: 1 (kill stalled processes) 0 (do not kill stalled processes) Default is 1. If you set this value to zero, you may want to implement a server extension to handle the stalled process on your own, in a customized way. The class `RunTimeApplication` has a function named `u_UserHandleFaultyProcess` that you can override to implement your own logic to handle a stalled process. The function receives the name of the faulty process as a parameter.

[LogClasses]

The log classes described in this section log usage data. This is separate and distinct from debug logging.

Table 24. Jacadasv.ini: [LogClasses] section

Parameter	Description

`<Log Java Class Name>`	This parameter specifies if any of the following log classes are to be used: `SessionLog`, `SessionCountLog`, `XMLLog`, `XMLServer`.
	
	A value of 1 means yes, 0 means no. Default is 0.
	
	Note: This parameter only gives you the ability to use the log class. To actually enable the log class, you must create a `[<Log Java Class Name>]`section for the class in question, and include the `Enable` parameter in that section.
	
	Format: `<Name of the Java log class>=<value>`.
	
	Example:
	
	`SessionLog=1`
	`SessionCountLog=1`
	`XMLLog=`
	`XMLServer=;`

[<Log Java Class Name>]

This section is used to set parameters for the log classes you chose in the `[LogClasses]` section. For each log class that you set to 1 in the `[LogClasses]` section, you must create a separate `[<Log Java Class Name>]`section. The four possible log classes are `SessionLog`, `SessionCountLog`, `XMLLog`, and `XMLServer`.

Table 25. Jacadasv.ini: [Log Java Class Name] section (Sheet 1 of 2)

Parameter	Description
`SessionLog`	The SessionLog writes the session status information to a log file in the format of a comma delimited list. For each active session, the SessionLog contains information such as: Server Id, Session Id, Time Connected, Time Disconnected, User Name, Profile, User Address, Application Name, Last Transaction, Total Transactions, Total Duration, Net Avg Duration, Avg Since Reset, Library Name, Current Screen, Last Event.
`SessionCountLog`	The SessionCountLog provides statistics about sessions running on the server. The following is a typical log entry: `[10/16/2000 at 16:46:00] Session count: 3, max: 3, avg: 1.55` **`count`** - number of active sessions. **`max`** - highest number of active sessions seen since logging began. **`avg`** - the average number of active sessions for the entire logged session.
`XMLLog`	The XMLLog class writes status information about the entire server system to a log file in standard XML format. The XMLLog includes information about the number, identity, and status of active processes, sessions, and applications; Server address, RMI port number, status of the processes, active applications.

Table 25. Jacadasv.ini: [Log Java Class Name] section (Sheet 2 of 2)

Parameter	Description
`XMLServer`	The XMLServer receives the same status information that is written to a file by the XMLLog class. The XMLServer makes the status information available for an online connection to the JIS Administrator.

[SessionLog]

If you set `SessionLog` to 1 in the LogClasses section, you must include the `[SessionLog]` section in the INI file.

Table 26. Jacadasv.ini: [SessionLog] section (Sheet 1 of 2)

Parameter	Description
`TimerTick`	Interval in seconds at which the Session log file is updated. Default is 3600 seconds (1 hour). The Server waits for this interval to elapse before writing its first log records to disk. Subsequent writes to the log file are also spaced at this interval. During the interval between writes, any new log information is buffered in memory, so log information is not lost. **Note**: If the JIS Server is closed down before the first TimerTick interval has elapsed, no Session log is created.
`Enable`	Specifies whether or not to enable Session logging. 0 = disable Session logging 1 = enable Session logging Default is 0.
`File`	The full pathname of the log output file. This parameter is required only when logging is enabled. If this parameter is omitted, the log file is not created. *Example:* `File=C:\temp\Session.log` webMethods JIS automatically inserts a timestamp in the file name. For example, if you specify `Session.log` as the file name and a FileInterval of 1d, the log file is created with the name `Sessionyymmdd.log`, where `yymmdd` is the date of the file's creation.

Table 26. Jacadasv.ini: [SessionLog] section (Sheet 2 of 2)

Parameter	Description
`FileInterval`	The interval for periodic creation of new Session log files. Supported time codes are: s - second, m - minute, h - hour, d - day. Precede the time code with an integer. *Example:* `12h`. Default value is 1d. When the specified time has elapsed, the existing Session logfile is closed and a new Session logfile is created and opened automatically. For example, if you specify a file interval of 2h, a new Session log file is created every two hours. The old file is not deleted. Duplicate filenames are avoided by the timestamp which is automatically added to the file name.

[SessionCountLog]

If you set `SessionCountLog` to 1 in the LogClasses section, you must include the `[SessionCountLog]` section in the INI file.

Table 27. Jacadasv.ini: [SessionCountLog] section (Sheet 1 of 2)

Parameter	Description
`TimerTick`	Interval for updating the SessionCount log file. Default is 3600 seconds (1 hour). The Server waits for this interval to elapse before writing its first log records to disk. Subsequent writes to the log file are also spaced at this interval. During the interval between writes, any new log information is preserved in memory, so no log information is lost. **Note**: If the JIS Server is closed down before the first TimerTick interval has elapsed, no SessionCount log is created.
`Enable`	Specifies whether or not to enable SessionCount logging. 0 = disable SessionCount logging 1 = enable SessionCount logging Default is 0.
`File`	The full pathname of the SessionCount log output file. This parameter is required only when SessionCount logging is enabled. If this parameter is omitted, the SessionCount log file is not created. *Example:* `File=C:\temp\SessionCount.log` webMethods JIS automatically inserts a timestamp in the file name. For example, if you specify `SessCt.log` as the file name and a FileInterval of 1d, the log file is created with the name `SessCtyymmdd.log`, where `yymmdd` is the date of the file's creation.

Table 27. Jacadasv.ini: [SessionCountLog] section (Sheet 2 of 2)

Parameter	Description
`FileInterval`	The interval for creating new SessionCount log files. Supported time codes are: s - second, m - minute, h - hour, d - day. Precede the time code with an integer. *Example:* `12h`. Default value is 1d. When the specified time has elapsed, a new SessionCount logfile is created automatically. For example, if you specify a file interval of 2h, a new SessionCount log file is created every two hours. The old file is not deleted. Duplicate filenames are avoided by the timestamp which is automatically added to the file name.

[XMLLog]

If you set `XMLLog` to 1 in the LogClasses section, you must include the `[XMLLog]` section in the INI file.

Table 28. Jacadasv.ini: [XMLLog] section

Parameter	Description
`TimerTick`	Interval in seconds at which the XML log file is updated. Default: 60.
`Enable`	Determines whether or not the XML log file is created. Valid values are 1 (yes) and 0 (no). Default: 0.
`File`	The pathname of the log output file. The setting is required only when logging is enabled. *Example:* `C:\temp\XMLlog.xml`

[XMLServer]

If you set `XMLServer` to 1 in the LogClasses section, you must include the `[XMLServer]` section in the INI file.

Table 29. Jacadasv.ini: [XMLServer] section

Parameter	Description
`TimerTick`	Refresh interval of the XML data that is sent to the JIS Administrator. Default is 60 seconds.
`Enable`	Defines whether or not the XMLLog data is sent to the JIS Administrator. Valid values are 1 (yes) and 0 (no). Default: 0. If you want to use the JIS Administrator, this parameter must be set to 1.

Scalability

This section provides you with information required to implement a scalable JIS Server system that dynamically balances the load in response to runtime demands.

Scalability provides you with a mechanism for creating a JIS Server system that can comprise a single server computer or a cluster of computers linked in a network. A single server computer can run several server processes. Any single process can handle multiple client/host sessions. The JIS Server system responds to client requests by dynamically opening and closing sessions on the server processes. The Scalability feature also provides a means for load balancing across the JIS Server system.

The Scalable System Structure of the JIS Server

The scalable JIS Server can consist of a single server computer or a series of server computers arranged in a network. In either case, the system derives its scalable nature from a structure of hierarchically arranged units called server modules. This section describes the position and relative function of server modules within the server system. During runtime a server module functions as a process. In terms of the hierarchical structure of the system each server module is uniquely identified by a server ID.

In this way each process is represented and identified as a node within the server system. First we describe the structure and terminology used for a server system that uses a single server computer. Second we describe a multiple server-computer system.

The hierarchical and functional relationship between the server processes, and if more than one server computer is used, between the server computers, is defined in the `jacadasv.ini` file. This is discussed in “Setting up the Scalable Server System” on page 145. Each server computer has its own `jacadasv.ini` file.

Single Server-Computer System

You can best understand the scalable nature of the JIS Server system if you first look at the structure of a simple system, one that consists of a single server computer with a limited number of server modules. Each process is represented by a node within the server system.

Structure

Figure 31 shows a single server computer that has three open processes. The highest level process on a server computer is referred to as the main node. Here the main node has two lower level nodes that are connected to it. These are referred to as sub-nodes.

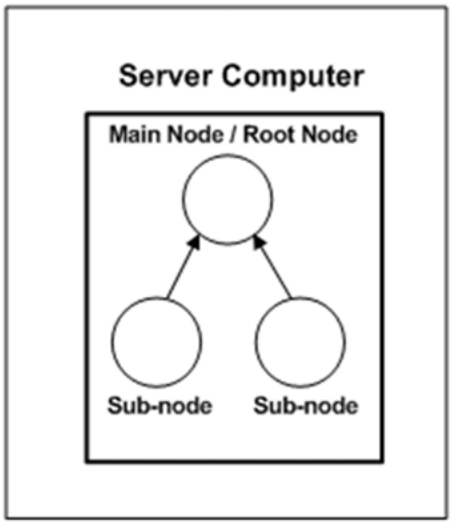


Figure 31. Single server computer system

Function

The simple structure of the system used in this scenario imposes a number of functional requirements on the main node. Functionally, when the lower level processes are first created it is the main node that is responsible for creating them. Any node that has sub-nodes attached to it is referred to as a parent node. Therefore in addition to being the structurally highest level node on the server computer, the main node also functions as the parent node of the two sub nodes. At set intervals each sub-node sends its status information to its parent node.

Status information updates include information such as:

- Whether the process is active.
- The number of host applications initialized by a process.
- The number of client/host sessions running.
- The number of client/host sessions that can run on a process.

In this way the parent node maintains status information on each of its sub-nodes.

This information is used to:

- Scale the system.
- Perform load balancing on the system.

In this scenario, the main node is also the highest level parent node in the entire system. The parent node in this position is referred to as the root node. The root node for any system maintains the most inclusive status information base for the server system. This node is the primary target of a client request for connection to a host application.

Client Connection to the System

In Figure 32, a client directs an initial request for a host connection to the root node. The root node has the most inclusive status information for the system, so it is the node that determines which process can most readily open a session with the host. It sends a message to the client, directing the client to send a second request for a host connection to a particular sub-node. The client sends a request to the particular sub-node and the process identified with the sub-node opens a client session with the host application.

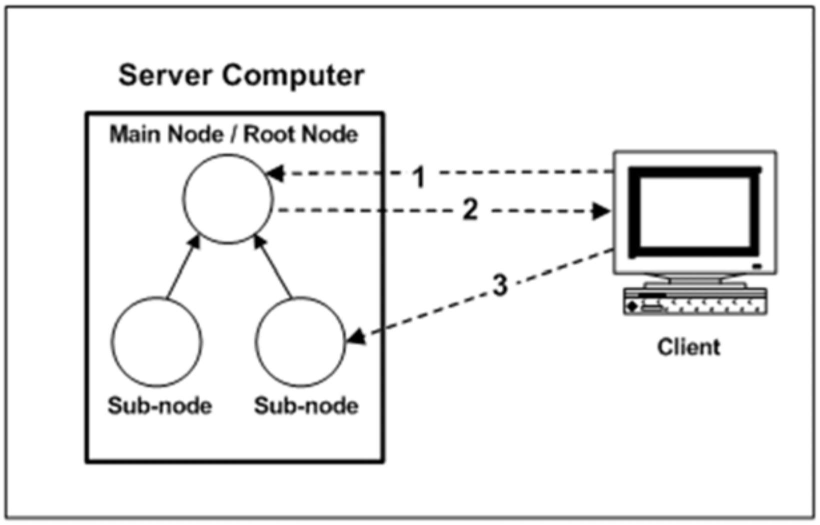


Figure 32. Control flow upon session start

Multiple Server-Computer System

The multiple server-computer system—also called a *server farm*—is organized as a hierarchically arranged group of server computers linked in a network. The hierarchical and functional relationship between server computers is defined in the `jacadasv.ini` file. See “Setting up the Scalable Server System” on page 145.

Structure

The illustration shows four server-computers, each with an identical internal process structure. While structurally an apparent similarity exists between the main nodes and the parent nodes, they differ functionally with respect to the position a server computer holds within the network.

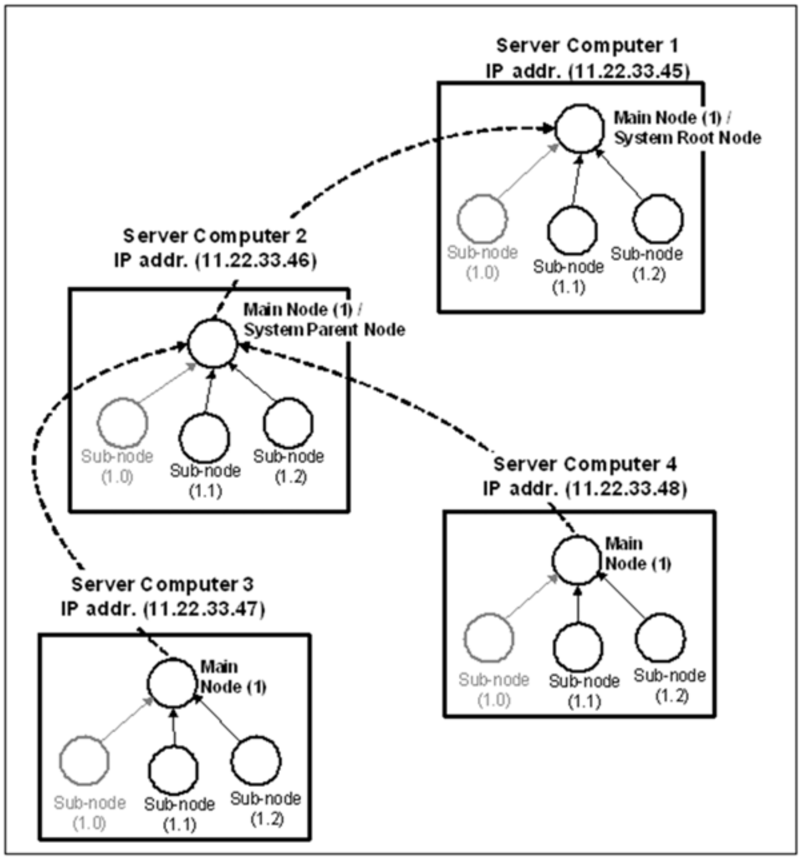


Figure 33. Multiple-server computer system

Each node in the server system is uniquely identifiable by a combination of the IP address of the computer it resides on and the node number, shown in parenthesis in Figure 33.

Function

The main node on any given server-computer is responsible for:

- Maintaining status information on each of the processes running on the computer.

- Transferring status information on each of its lower-level processes, to its designated system parent-node.
- Initializing lower level processes on that particular computer.

The functional difference between main nodes in a multiple server-computer system depends on the position the server-computer holds within the network.

In the scenario presented in Figure 33, server-computers 3 and 4 are the lowest level server computers. Each of their main-nodes maintains data on its sub-nodes and transfers the information to the main node on server-computer 2.

Any main node that receives status information from lower level main nodes is referred to as a system parent-node. In Figure 33 the main node on server-computer 2 is acting as the system parent-node.

Server-computer 2 maintains information on its own processes and those of server-computers 3 and 4, and transfers this information to server-computer 1's main node. This being the highest level server computer, its main-node functions as the root-node. The root node maintains the most inclusive amount of information available from the system.

You can see from the scenario provided that status information is transferred from the lowest level in the hierarchy to the highest level, exclusively via main nodes.

Note: The hierarchical structure presented in the example is designed to illustrate system elements. The structure of a working multiple server computer system is flexible and should be designed to best support your hardware, application and client needs.

Client Connection to the System

The process here is slightly different to that described for a single server computer system. Again the client directs its initial request for a host connection to the root node. Again the root node has the most inclusive status information for the system; however, in this case it determines which server computer has the least load and re-directs the client request for a host connection to a sub-node of the computer server that presents the least load. The sub-node then takes responsibility for opening a client session.

If server computer 4 presents the least load, then the following occurs:

- The client directs its initial host connection request to the root-node on server-computer 1.
- The root-node examines its status information and determines that server-computer 4 has the least load.
- Server-computer 1 redirects the client request to the process on server-computer 4's main that carries the least load.

This example describes the general flow of events that occurs when a client-host connection request is redirected from the main server machine to a secondary server machine in the system.

Identifying Server Modules

During runtime, Server modules act as processes. Each process can be identified from any other process on a server computer by a combination of the IP address of the machine it resides on and its process ID. Process IDs can be the same on different machines, so the IP address is needed to differentiate them.

For example, in Figure 33, the ID of the main node on server-computer 4 is 11.22.33.48.1, and 11.22.33.48.1.1 is one of its sub-nodes.

The server ID indicates:

- The hierarchical level that the process belongs to on the server computer.
- The position that the process holds within the level.

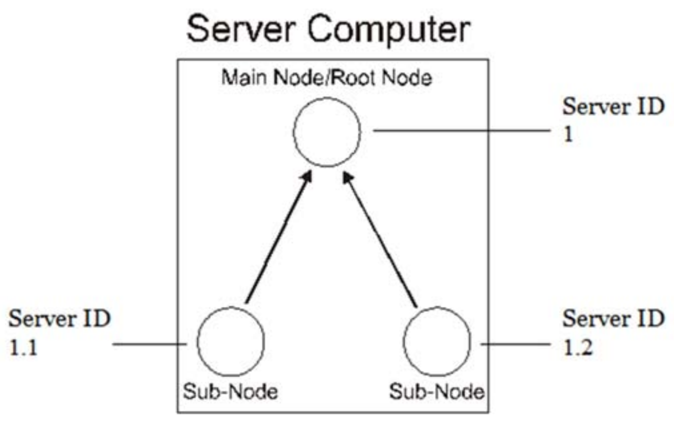


Figure 34. Identifying server modules

The root node is the only node on the highest level and so its server ID is 1. The root node has created two sub-nodes so their processes are identified as 1.1 and 1.2. The existence of a second digit indicates that processes are located on the second level of the hierarchy, and the value of the second digit indicates the position held within the level. If the sub-node 1.2 was directed to create a sub-node of its own then the first process on the third level would be created and its server ID would be 1.2.1.

Note: This naming system exists within a server computer and not across server computers.

The Integrator Process

In Figure 33, note that each Server machine includes a sub-node numbered 1.0. These nodes are known as *integrator processes*. At server start-up, each integrator process is responsible for the creation (or *spawning*) of all processes on the machine on which it resides, based on the decisions of the main node.

When dynamic process spawning is used, each main node analyzes the load on its own machine and instructs the integrator process to spawn new processes as needed.

An integrator process is automatically created on each server machine that has more than one level of processes defined. Software AG generally recommends that the integrator process not handle any client sessions itself, so that it is devoted to load processing. Do this by including the following lines in the `jacadasv.ini` file for each server machine.

```
[<Machine>.Integrator.Sessions]
MaxProcessSessions=0
```

Setting up the Scalable Server System

To set up the scalable system of JIS Server:

1. Perform the runtime installation process on each machine that functions as a server computer for the system.
2. Follow all regular server setup procedures, as indicated in the manual. This must be performed for each machine functioning as a server computer.
3. Replace the `jacadasv.ini` file supplied during runtime installation with the customized `jacadasv.ini` file produced for running the scalable server system.

Note: Be aware that the `jacadasv.ini` file produced at runtime generation contains an `[Applications]` section which contains information about the runtime Applications. A section for each Application listed in the `[Applications]` section is also created in the `jacadasv.ini` file at runtime generation. These sections must be reflected in the customized `jacadasv.ini` file produced for running the scalable server system.

Customized jacadasv.ini File

A customized version of the `jacadasv.ini` is used when running a scalable server system, which contains different information to the regular `jacadasv.ini` that is supplied as part of the runtime installation. This special `jacadasv.ini` contains specific parameters which are used by the scalability feature.

These parameters are used for:

- Defining the startup state of the system.
- Indicating how the system is to behave, with respect to scaling and load-balancing, in response to client requests to open a host session.

The contents of the customized `jacadasv.ini` file reflect the hardware, application and client requirements that best support your needs. To gain the best results from your system we recommend you customize the `jacadasv.ini` file in consultation with a Software AG representative.

The following sections provide:

- A description of the general structure of a customized `jacadasv.ini` file.
- Examples of the `jacadasv.ini` file for single server-computer and multiple server-computer systems.

General Structure of the jacadasv.ini File

The `jacadasv.ini` file provides a generic layout of the server system to be constructed. When a process is started it looks up the information that belongs to its hierarchical level and task in the system, according to its local address and the identification it received on the command line. A unique initialization set is provided for each server computer and for each node's hierarchy level on the server computer. Parameters defined in one section can be overridden by parameters set in another section according to certain rules, explained below.

The jacadasv.ini File is Composed of Sections

The `jacadasv.ini` file is composed of sections. Each section is headed by a line with the section name in square brackets; for example:

```
[Sessions]
```

Under the section header are one or more lines of parameters. For a complete listing of all of the section names and their parameters, see "JIS Server INI File Settings" on page 105.

Targeting ini Parameters to a Particular Machine or Server-Node Level

It is possible to selectively target the parameters in a `jacadasv.ini` file section to a particular machine or even to a particular node-level within a specific machine. This is done by specifying additional information in the section header line. The general format is

`<Machine>.<Section>` or `<Machine>.<NodeLevel>.<Section>`

If you use the format `<Machine>.<Section>` in a `jacadasv.ini` file section header, the parameters in that section apply only to the machine specified in the header.

If you use the format `<Machine>.<NodeLevel>.<Section>` in a `jacadasv.ini` file section header, the parameters in that section apply only to the machine and node-level combination specified.

Permitted values for `<Machine>` are determined by the machine names you specify in the `jacadasv.ini` file `[ServerMachines]` section.

Permitted values for `<Level>` are:

- **`Level1`** - for the main node.
- **`Integrator`** - for the integrator node.
- **`Level2`** - for the nodes immediately subordinate to the main node (except the integrator node).
- **`Level3`** - for the nodes immediately subordinate to Level2 nodes, and so on.

Examples

The sample jacadasv.ini file shown in Example 2, "jacadasv.ini file for a Multiple Server-Computer System", includes several file sections related to Sessions parameters:

- The section headed **`[M1.Level1.Sessions]`** affects only the main node (always node 1) on machine M1. (The identity of computer M1 is defined in the `[ServerMachines]` section.)
- The section labelled **`[M1.Integrator.Sessions]`** applies only to the integrator node (always node 1.0) on machine M1.
- The section headed **`[M2.Level1.Sessions]`** affects only the main node (always node 1) on machine M2.
- The section labelled **`[M2.Integrator.Sessions]`** applies only to the integrator node (always node 1.0) on machine M2.
- The section labelled **`[M1.Sessions]`** applies to all the nodes on machine M1, except the nodes specifically targeted by other sections.

Precedence of Targeted ini File Sections

The general rule is: for a given node or node-level, the settings under a more specific section header take precedence over the settings under a less specific section header.

Example: In a multiple server-computer environment, you could have a `jacadasv.ini` file with three different types of `[GeneralParameters]` section headings:

```
[M2.Level2.GeneralParameters]
[M2.GeneralParameters]
[GeneralParameters]
```

In such a case, the section headed `[GeneralParameters]` applies to all nodes, with the following exceptions:

- On machine 2, any settings in the `[M2.GeneralParameters]` section take precedence over the settings in the `[GeneralParameters]` section.
- For the Level2 nodes on machine M2, the parameter settings in the section labelled `[M2.Level2.GeneralParameters]` take precedence over the settings in the `[M2.GeneralParameters]` and `[GeneralParameters]` sections.

Example 1. jacadasv.ini for a Single Server-Computer System

```
[GeneralParameters]
RtRootDir=i:\java\
HTTPClient=1

[Xhtml]
RuntimeDirectory=$RootDir\classes\appls
ImagesLocation=/classes/appls
DoHTMLMerge=1

[HTTP]
HTTPPortRange=8081-8180
HTTPSPortRange=8152-8250
SupportHTTPS=0
ResourceBase=I:\java\

[ServerMachines]
//The addresses of the server machines are defined here. “M1” stands for
//“machine 1”. It’s an arbitrary convention for distinguishing one server
//computer from another. This setup uses only one server computer.
```

```
10.11.12.101=M1

[M1.Level1.HTTP]
HTTPPortRange=8080-8080
HTTPSPortRange=8151-8151
//If using HTTP proxy, the HTTPSPortRange parm
// should be in the M1.Integrator.HTTP section.

[M1.GeneralParameters]
//This machine will handle a maximum of 12 processes.
MaxProcesses=12

[M1.Level1.Sessions]
//The root node will not process client sessions.
MaxProcessSessions=0

[M1.Integrator.Sessions]
//The integrator node will not process client sessions.
MaxProcessSessions=0

[Sessions]
//A maximum of 600 total sessions will be created.
//A maximum of 12 total processes were defined above, in the
//GeneralParameters section. One of those processes will be the root node,
//another will be the integrator process. Those two processes were defined
//as handling zero client sessions. That leaves 10 processes to handle
//client sessions. Each of the 10 processes are define as handling up to
//60 sessions,for a maximum of 600 client sessions.
StartUpSessionsPercent=100
SpareSessionsPercent=0
MaxProcessSessions=60
MaxMachineSessions=600

[LogClasses]
XMLServer=1

[XMLServer]
Enable=1
TimerTick=20

[M1.VMCommandLine.NodeRegistry]
JavaMemory=-ms50m -mx100m
```

```
[M1.VMCommandLine.Server]
JavaMemory=-ms300m -mx600m

[M1.Integrator.VMCommandLine.Server]
JavaMemory=-ms100m -mx200m

[M1.Level1.VMCommandLine.Server]
JavaMemory=-ms50m -mx100m

[Applications]
PRODAP01=

[LOADTEST]
WorkingDirectory=$RootDir\classes\appls\LOADTEST\server\resources\
IniDir=$RootDir\classes\appls\LOADTEST\server\resources\
```

Example 2. jacadasv.ini file for a Multiple Server-Computer System

```
[GeneralParameters]
RtRootDir=i:\java\
HTTPClient=1

[Xhtml]
RuntimeDirectory=$RootDir\classes\appls
ImagesLocation=/classes/appls
DoHTMLMerge=1

[ServerMachines]
//The addresses of the server machines are defined here. “M1” stands for
//“machine 1”, “M2” is “machine 2”. It’s an arbitrary convention for
//distinguishing one server machine from another. This setup uses 2 servers.
10.11.12.105=M1
10.11.12.110=M2

[M1.GeneralParameters]
MaxProcesses=12

[M2.GeneralParameters]
MaxProcesses=4
```

```
[M2.GeneralParameters]
ReportsToMachine=M1

[M1.Level1.Sessions]
//The main node on machine 1 will not process client sessions.
MaxProcessSessions=0

[M1.Integrator.Sessions]
//The integrator node on machine 1 will not process client sessions.
MaxProcessSessions=0

[M2.Level1.Sessions]
//The main node on machine 2 will not process client sessions.
MaxProcessSessions=0

[M2.Integrator.Sessions]
//The integrator node on machine 2 will not process client sessions.
MaxProcessSessions=0

[M1.Sessions]
StartUpSessionsPercent=100
SpareSessionsPercent=0
MaxProcessSessions=60
MaxMachineSessions=600

[M2.Sessions]
StartUpSessionsPercent=100
SpareSessionsPercent=0
MaxProcessSessions=20
MaxMachineSessions=50

[M1.Level1.HTTP]
HTTPPortRange=8080-8080
HTTPSPortRange=8151-8151

[M2.Level1.HTTP]
HTTPPortRange=8090-8090
HTTPSPortRange=8161-8161

[M1.HTTP]
HTTPPortRange=8081-8180
```

```
HTTPSPortRange=8152-8250
ResourceBase=I:\java\

[M2.HTTP]
HTTPPortRange=8091-8190
HTTPSPortRange=8162-8260
ResourceBase=I:\java\

[LogClasses]
XMLServer=

[XMLServer]
Enable=1
TimerTick=20

[M1.VMCommandLine.NodeRegistry]
JavaMemory=-ms50m -mx100m

[M1.VMCommandLine.Server]
JavaMemory=-ms300m -mx600m

[M1.Integrator.VMCommandLine.Server]
JavaMemory=-ms100m -mx200m

[M1.Level1.VMCommandLine.Server]
JavaMemory=-ms50m -mx100m

[M2.VMCommandLine.NodeRegistry]
JavaMemory=-ms50m -mx100m

[M2.VMCommandLine.Server]
JavaMemory=-ms300m -mx600m

[M2.Integrator.VMCommandLine.Server]
JavaMemory=-ms100m -mx200m

[M2.Level1.VMCommandLine.Server]
JavaMemory=-ms50m -mx100m

[Applications]
PRODAP01=
[PRODAP01]
WorkingDirectory=i:\java\classes\appls\PRODAP01\
server\resources\
IniDir=I:\guisysd\appls/PRODAP01\rt32\
```

Redirection Proxy

The redirection proxy handles redirections of connections sent from the JIS Server. When using the redirection proxy, the end-user is unaware of the redirection event. You can use the redirection proxy to bar end-users from gaining access to your IP address and port numbers.

The Redirection Process

The redirection proxy runs as a part of the JIS Server integrator process.

The redirection process is as follows:

1 The end-user sends a request to the proxy, instead of the JIS Server.
2 The proxy connects to the JIS Server’s IP address and handles redirection events, if any, from the JIS Server.
3 The proxy encrypts the address of the redirected machine and port and appends it to the URL.

In subsequent requests, the end-user’s browser displays the encrypted JIS Server’s IP address as the JBS parameter concatenated to the URL.

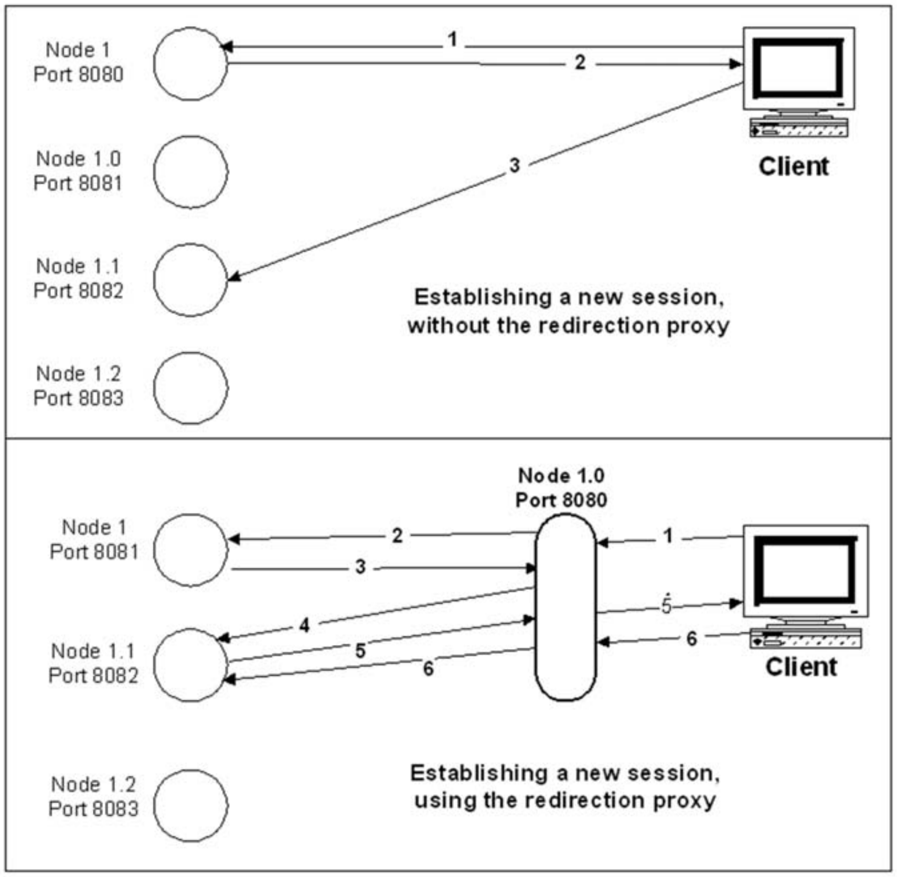


Figure 35. Redirection proxy mechanism

Starting the Redirection Process

To start the redirection process:

1 Configure the `jacadasv.ini` file to enable the redirection process.

2 Configure the proxy configuration file.

Configuring the jacadasv.ini File to Enable the Redirection Process

To configure the jacadasv.ini file to enable the redirection process:

Insert the following settings into the `jacadasv.ini` file:

```
[M1.Integrator.HTTP]
HTTPPortRange=8080-8080
JettyConfigurationFile=proxyHttp.xml
ProxyConfigurationFile=$RootDir\classes\proxyconfiguration.xml
```

This configures the M1 integrator process's HTTP settings as follows:

- The `HTTPPortRange` parameter indicates that the process runs the proxy on port 8080.
- The `JettyConfigurationFile` parameter setting indicates that the Jetty HTTP server loads the `proxyHttp.xml` configuration file. This file tells Jetty to run the redirection proxy.
- The `ProxyConfigurationFile` setting assigns the proxy configuration file name: `proxyconfiguration.xml`.

Configuring the Proxy Configuration File

The redirection proxy is a servlet, and is configured by the `proxyconfiguration.xml` file.

The `proxyconfiguration.xml` file looks like this:

```
<Settings>
   <JacadaServerAddress>
      <IPAddress>localhost</IPAddress>
      <Port>8081</Port>
   </JacadaServerAddress>
   <DebugInfo>
      <LogPath>c:\temp</LogPath>
      <DebugLevel>50</DebugLevel>
   </DebugInfo>
       <RedirectionProxy>
      <RunProxyOnEachMachine>Yes</     <RunProxyOnEachMachine>
      <ProxyPort>8080</ProxyPort>
      <ProxyHTTPSPort>8443</ProxyHTTPSPort>
      <UseGzip>No</UseGzip>
      <CloseStreamToClient>No</CloseStreamToClient>
       </RedirectionProxy>
</Settings>
```

Settings in the proxyconfiguration.xml file

The `proxyconfiguration.xml` file contains the following settings:

`<JacadaServerAddress>`

This tag is used for defining the JIS Server root process address.

The `<JacadaServerAddress>` tag contains the following tags:

- The `<IPAddress>` tag defines the JIS Server's IP address.
- The `<Port>` tag defines the port number of the rootprocess (level1) of the server machine.

`<DebugInfo>`

Note: The <DebugInfo> tag is now deprecated. By default, the redirection proxy will log messages to the integrator process log file debug_1.0.log

This tag is for defining the proxy's debug information settings.

The `<DebugInfo>` tag contains the following tags:

- The `<LogPath>` tag defines the path to the log file.
- The `<LogFileName>` tag is no longer relevant and is ignored if present. See "The Proxy Log File" on page 157 for more information.
- The `<DebugLevel>` tag defines the debug level.

`<RedirectionProxy>`

This tag is used for running the proxy on each of your machines.

The `<RedirectionProxy>` tag contains the following settings:

- The `<RunProxyOnEachMachine>` tag specifies whether or not run the proxy on each of your machines. The tag's value can be `YES` or `NO`.
- The `<ProxyPort>` tag defines the port on which all proxies listen.
- The `<ProxyHTTPSPort>` tag defines the port which listens for HTTPS requests.
- The `<UseGzip>` tag controls whether or not the redirection proxy can send GZIP compressed content to the browser. The proxy sends GZIP only if the browser sends an HTTP header "accept-encoding" that contains the value "gzip" inside and the setting specifies that GZIP compression is to be used. Most browsers support GZIP compression.
- The `<CloseStreamToClient>` tag is discussed in the following section.

Using a Load Balancer or Another Proxy with the JIS Redirection Proxy

In some customer environments, an additional proxy or a load balancer may be present in front of webMethods JIS's redirection proxy. A small number of customers with this arrangement have reported problems related to the fact that the webMethods JIS redirection proxy closes the stream to the client after handling the client HTTP request.

The problem occurs if the additional proxy or load balancer expects the stream to remain open for a certain period of time after the completion of the request.

To address this (somewhat rare) issue, a setting has been added to the webMethods JIS `proxyConfiguration.xml` to give control over whether or not the stream to the client is closed by the JIS proxy after the HTTP request has been handled.

The setting is `CloseStreamToClient`, in the `RedirectionProxy` section of the `proxyConfiguration.xml` file. Setting `CloseStreamToClient` to `No` instructs the JIS proxy to leave the stream to the client open. The default behavior is to close the stream.

Here is an example of a configuration:

```
<Settings>
   ...
   <RedirectionProxy>
      ...
      <CloseStreamToClient>No</CloseStreamToClient>
   ...
   </RedirectionProxy>
</Settings>
```

When `CloseStreamToClient` stream is set to No, at debug level 50 a message is written to the proxy log stating: `Not closing stream to client`.

The Proxy Log File

As of webMethods JIS version 9.0, when running the webMethods JIS proprietary server, the proxy runs inside the integrator process (1.0) and the proxy's debug messages are written to `debug_1.0.log`. There is no longer a separate proxy log when running with the JIS Server.

The old setting `LogFileName` in the `proxyConfiguration.xml` file is no longer relevant and if present is ignored.

Running a Proxy on Each of your Machines

You can run a proxy on each machine for scalability. When doing so, the end-user still connects to the root machine proxy. If redirected to a different machine, then the end-user is redirected to that machine's proxy. The URL appearing on the end-user's browser also changes to the specific machine's proxy.

To run the proxy on each of the machines:

Set the `jacadasv.ini` file as follows:

```
[M1.Integrator.HTTP]
HTTPPortRange=8080-8080
JettyConfigurationFile=proxyHttp.xml
ProxyConfigurationFile=$RootDir\classes\proxyconfiguration.xml
[M2.Integrator.HTTP]
HTTPPortRange=8080-8080
JettyConfigurationFile=proxyHttp.xml
ProxyConfigurationFile=$RootDir\classes\proxyconfiguration.xml
```

```
[M3.Integrator.HTTP]
HTTPPortRange=8080-8080
JettyConfigurationFile=proxyHttp.xml
ProxyConfigurationFile=$RootDir\classes\proxyconfiguration.xml
```

Note: All proxies must run on the same port. You must indicate this in both in the jacadasv.ini file (via the `HTTPPortRange` parameter) and in the `proxyconfiguration.xml` file (via the `ProxyPort` tag).

HTTPS Communication Described

HTTPS is a means of ensuring secure communication.

HTTPS works through the use of certificates. A certificate is a file that has been digitally signed by a Certificate Authority, and contains information about the web server. The browser can check that the information and the digital signature are correct, provided that it recognizes the Certificate Authority that has issued the certificate.

In order to get a real certificate, you must first create a certificate signing request and then send that request to a Certificate Authority, who will then check that you are who you claim to be, and then give you a certificate that they have digitally signed. The process can take several days or more, so plan accordingly.

Using HTTPS Communication

You enable HTTPS slightly differently, depending on whether or not you are using the redirection proxy.

To enable HTTPS communication when not using the redirection proxy:

1 In the `jacadasv.ini` file's `[<Machine>.Level1.HTTP]` section, insert an `HTTPSPortRange` parameter in addition to the existing `HTTPPortRange` parameter. Provide a different port number to the one assigned for HTTP.

Example: `HTTPSPortRange=8443-8443`

2 In the same `jacadasv.ini` file section, insert the parameter `HTTPSSupport=1`

To enable HTTPS communication when using the redirection proxy:

1 In the `jacadasv.ini` file's `[<Machine>.Integrator.HTTP]` section, insert an `HTTPSPortRange` parameter in addition to the existing `HTTPPortRange` parameter. Provide a different port number to the one assigned for HTTP.

Example: `HTTPSPortRange=8443-8443`

2 In the same `jacadasv.ini` file section, insert the parameter `HTTPSSupport=1`

3 In the `proxyConfiguration.xml` file, set the `<ProxyHTTPSPort>` tag's value to the same port number set in the `jacadasv.ini` file.

Example: `<ProxyHTTPSPort>8443</ProxyHTTPSPort>`

JIS Server Logging Support

This section introduces the JIS Server mechanism for tracking and viewing the session status information for each process on your server system. Periodically this information is written to a log file that is then available for you to read.

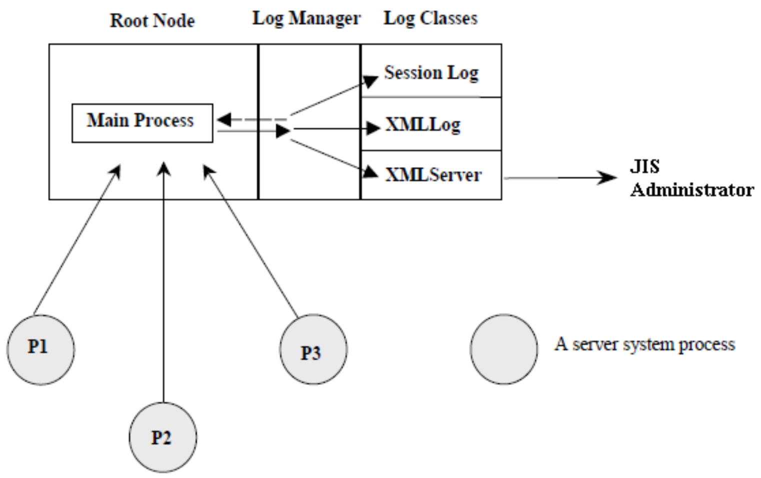


Figure 36. JIS Server logging architecture

JIS Server Logging Architecture

The JIS Server logging structure is composed of several units that communicate with each other and with the server system's root node. These units include the Log Manager and each of three Log Classes; the Session Log, the XMLLog and the XMLServer.

The following table lists and explains the Server System's logging support components:

Table 30. JIS Server logging system

Component	Function
Log Manager	The Log Manager is responsible for storing session status information. Information held by the Log Manager is periodically updated in response to log class requests.
SessionLog	The SessionLog writes the session status information to a log file in the format of a comma delimited list. The log file location is read from the SessionLog section in the `jacadasv.ini` file. Status information written to the SessionLog is limited to specific data parameters for each session.
XMLLog	The XMLLog writes the status information to a log file in standard XML format. The log file location is indicated in the XMLLog section of the `jacadasv.ini` file. The status information written by the XMLLog class includes a record of the complete status information across the entire server system.
XMLServer	The XMLServer receives the same set of status information as that for the XMLLog class. Instead of writing the information to a log file, the XMLServer makes the status information available for an online connection to the JIS Administrator.
JIS Administrator	When connected to the server system's root node the JIS Administrator provides a user interface for viewing the XMLServer output online, and by remote access.

JIS Server Log Information Flow

Session status information is directed to the root node in response to two different types of events:

- When a process opens or closes on a server module.
  -OR-
- When there is a log class request.

When a process either opens or closes, its session status information is sent to the root node. This information is then available to the Log Manager. In addition, the Log Manager responds to periodic Log Class requests for session status information by instructing the processes to send session status information. The type of information sent to root node is dependent on the Log class performing the request.

The Server System Log Classes

SessionLog Log Class

The SessionLog log class writes session status information to a log file in the format of a comma delimited list. The first record in the list is a set of column headers. Each remaining record contains data associated with each of the column headers. Each record in the list represents information about a particular session on a particular process.

Processes record information concerning any one of their sessions when:

- A session opens
- A session closes
- A screen changes
- A session command event occurs

In this way the information written to the session log forms an incremental record of the information and events occurring on processes and their sessions.

Table 31 indicates the record parameters written to the SessionLog and provides a description of each parameter:

Table 31. SessionLog records (Sheet 1 of 2)

Parameter	Description
`Server Process ID`	The server IP address and Port1. Separated by a colon (:). *Example*: `192.90.14.4: 1100`
`Session ID`	A numeral representing the session's position in the process relative to any other session. Example: 10 indicates that the session is the tenth session to be opened by the process.
`Time Connected`	A time stamp indicating the date and time that the session connected to the process. The time is given in hours:minutes:seconds. *Example*: `2/18/99 12:19:40`
`Time Disconnected`	A time stamp indicating the date and time that the session was disconnected from the process. The time is given in hours:minutes:seconds. *Example*: `2/18/99 12:50:30`
`User Name`	The user name taken at login. `_Default_User_`
`User Profile`	User profile name
`User Address`	The Client address. `Localhost/127.0.0.3`
`Application`	The name of the Application running on the host. `MBF`
`Time of Last Transaction`	The time of the last recorded server activity in the current session.

Table 31. SessionLog records (Sheet 2 of 2)

Parameter	Description
`Total Transact Duration`	Time taken from request reception to the return of a response.
`Number of Transactions`	The total number of transactions performed by the process.
`Aver Transact Duration`	The average transaction duration - calculated by dividing `Total Transaction Duration` by `Number of Transactions`.
`Current Screen`	Provides the name of the current Subapplication.
`Session Event`	The session event for which the parameters were taken. `Session_New`

Viewing the SessionLog Output

The SessionLog output is written to a log file as defined in the `jacadasv.ini`. This is a text file that is best viewed by importing the contents into any spreadsheet or application that is capable of displaying comma delimited lists. For example, Excel, Access or Lotus 1-2-3.

Setting the LogClasses and Their jacadasv.ini File Parameters

The `jacadasv.ini` file contains a number of sections that are relevant to the Server Logging support feature: A section called `[LogClasses]` for defining the log classes that are available for use by the server, and a separate section for each specific log class. The specific log class sections define parameters specific for the logclass' operation.

These include:

- Whether the LogClass is enabled or disabled.
- The frequency at which a LogClass sends a request to update data.
- The path of the log file to which the LogClass writes data.

Table 32. LogClass parameters

INI Entry	Values and Description
`Enable`	Defines whether the LogClass is available for use by the server during runtime. 1 indicates that the LogClass is enabled. 0 indicates that the LogClass is disabled. In future versions this parameter will behave as a flag that can be set from a remote location.
`TimerTick`	Defines the amount of time between successive log class requests for data updates. This is given by a numeral that indicates time in seconds.
`FileInterval`	Defines the interval for creating new log files. The supported time codes are: **s -** second, **m -** minute, **h -** hour, **d -** day. *Example*: `12h`
`File`	Defines the location and file name to which the LogClass writes data.

LogClasses Section

You define the LogClasses for use by JIS Server logging support during runtime in the `[LogClasses]` section of the `jacadasv.ini` file. Each entry in the section represents a different LogClass. The section appears similar to this:

```
[LogClasses]
;XMLLog=
;XMLServer=
SessionLog=
```

Note: In the section illustrated, the XMLLog and XMLServer have been commented out by including a semi-colon (;) before the LogClass name. This is one method of disabling the use of the LogClass.

SessionLog Section

```
[SessionLog]
Enable=1
```

```
TimerTick=30
File=C:\log\Jacadasv.log
```

XMLLog Section

```
[XMLLog]
Enable=0
TimerTick=20
FileInterval=6h
File=C:\log\Jacadasv.xml
```

XMLServer Section

```
[XMLServer]
Enable=1
TimerTick=30
```

How to Create a Server Log File

To create a Server Log File, in the `jacadasv.bat` file, specify a debug level higher than zero on the batch statement; for example: `-d50`.

You can also name the log file and place it in a location of your choice, by specifying `-l`<drive and path>`\FileName`.

Alternately, you can specify the location of the logfiles by adding a parameter to the `[GeneralParameters]` section of the `jacadasv.ini` file:

`RtLogsDir=$RootDir\<directory>`

For example: `RtLogsDir=$RootDir\classes\logs`

Example 3. Setting up a Server Log file

```
c:\jis>jacadasv.bat -d70 -lc:\temp
```

Advanced Logging Features

This section discusses some advanced logging features:

- Controlling the absolute size of the server log file.
- Using filters to limit log output.

Controlling the Size of the Log File

The command line option `-m` lets you specify the maximum size of the log file, in bytes. Once the specified size is reached, wraparound occurs; that is, the log file is overwritten from the top.

The Start Log

During JIS Server startup, log messages are written describing the server environment, including the operating system version, JVM number, server build number, and INI file settings. This information can be very useful for debugging

These startup messages are written to a separate log file, named `debug_start.log`. They are written to a separate log file so that they are preserved even when logfile wraparound is in effect.

Debug Filters

Debug Filters are tools to help you accomplish specific types of logging. They are implemented by using the command line parameter

```
-f<Debug Log Filters (comma delimited)>
```

We recommend that the Debug Filters be used with a low debug level, even debug level 1 or 2; this makes it easy to find the filtered messages in the output. No filtered messages are printed, however, when the debug level is 0.

Example: `-fSCREEN_IMG_NL  -d2`

Activating Debug Filters While the JIS Server is Running

Debug filters can also be activated or deactivated while the JIS Server is running, without stopping and restarting the Server. This is done by entering a command in the server window.

`addfilter <FILTERNAME>` - Turns on the specified filter. Specify just one filter. If you want to turn on more than one filter, execute the `addfilter` command once for each filter desired.

If you specify no operand, the `addfilter` command lists on the JIS Server console the names of all existing predefined filters.

`removefilter <FILTERNAME>` -Turns off the specified filter. Specify just one filter. If you want to turn off more than one filter, execute the `removefilter` command once for each filter desired.

`displaycurrentfilters` - Lists to the JIS Server console the names of all filters currently in use.

Example: `addfilter SCREEN_IMG`

Printing the Host Screen

When debugging an Application, it can sometimes be useful to print the host screen. Debug filters are one way of accomplishing this.

webMethods JIS comes with three built-in Debug Filters that let you print the host screen. When a Debug Filter is turned on, it is in effect for all sessions, and for the complete lifetime of a session.

Following is a description of the debug filters for printing host screens.

Filter name: SCREEN_IMG This log filter prints the host screen as one long string, including attribute bytes.

Filter name: SCREEN_IMG_NL This log filter prints the host screen as seen from the client host view, in a rectangular format, including attribute bytes.

Filter name: SCREEN_IMG_PROG Prints the screen image in a rectangular format after every progress message that the server receives from the host. The screen image printed includes attributes.

Scalability Filter

Filter name: SCALABILITY The scalability filter provides log information that is relevant to load testing and server tuning, performance monitoring, and screen identification problems. Because the scalability filter is intended for use in servers running a large number of users, its output is limited to information essential to scalability debugging. Among the information displayed by the scalability filter is:

- new data arriving from the host.
- keyboard unlock events.
- screen identification results.
- `SetWaitForScreenState` information.

Method Debug Filter

Filter name: METHOD

The method debug filter limits output to the method debug messages. See "Setting Runtime Generation Options" on page 63 for more information about generating method debug messages. The method debug messages assist in the tracing and debugging of user-written methods.

Analyzing Abnormal Runtime Termination

To provide a means of analyzing abnormal runtime termination, specific information about session termination is written to logs called Dump Files.

Information Included in Dump Files

Dump files record events such as client disconnections, server exceptions, and host failure. The information in these log files is recorded just before the session closes.

Dump files contain three levels of information:

Table 33. Dump files data

Level	Description
General Process Level Information	This level includes information that stays the same for all sessions running under a given server process. This information is updated when the server process is started.
Session Level Information	This level includes session exceptions and information regarding the current state of a given session. Session level information is updated whenever the session state is changed. Each property in this section of the dump file includes a timestamp of the last update.
Exception Information	The first exception of each exception class is included in the data of the dump file.

Dump File Generation

The server generates dump files anytime a session closes without the user's intervention.

Dump File Name and Location

A separate dump file is created for each problematic session.

The server dump file is named: `jbs_<machine>_<process>_<session).log`

and is located in: `<InstallDir>\JacadaFiles\classes\logs`.

Enabling Dump File Generation

The behavior of dump files can be set in both the runtime *.ini file (`<applname>.ini`) and the `jacadasv.ini` file. Set the following parameters to enable dump file generation and determine the various logging properties:

[SessionCoreDump]

Table 34. Setting dump parameters

Parameter	Value	Description
`AlwaysDump`	0	Default value. Dump files are only generated on abnormal termination.
	1	Always generates dump files on the server, regardless of how the session ended.
`DumpFast`	1	Default value. Only saves information pertaining to abnormal termination.
	0	Additional information is saved, including detailed descriptions of the current Subapplication's emulator events and variables. This causes approximately 5% performance deterioration.

Dump File Structure

The dump files contain three types of information:

- General Process Level information.
- Session Level information.
- Exception information.

This section contains an example of a Java Client Core Dump file and a Session Core Dump file. Each example lists the type of information contained in the dump file and then illustrates the information in an extraction from a dump file.

Java Client Core Dump File

The Client Core Dump file includes the following details:

Table 35 lists the type of information found in the General Process information section.

Table 36 lists the type of information found in the Session Level information section.

Table 37 lists the type of information found in the Exception information section.

Table 35. Client dump: general process information (Sheet 1 of 2)

Information Type	Example
Java vendor and version number	Java vendor: Sun Microsystems Inc., version: 1.4.2_02
OS name and architecture number	OS name: Windows 2000, architecture: x86
JIS version or PTF name	Version/PTF Name = 8.1
JIS version Build number	BuildNumber = 8,1,0,54
Machine IP address	Machine IP address: 10.90.18.149
JVM memory consumption	Memory: Free: 49 Kb (3%) Total: 1488Kb

Table 35. Client dump: general process information (Sheet 2 of 2)

Information Type	Example
Session termination details	Session was terminated due to: 2/12/03 11:36:00 AM Quit reason: Exception in CommServer: java.lang.NullPointerException
Reference to a session log file	A matching dump file may have been created on server M1, named 10.90.18.149_1_1.log

Table 36. Client dump: session level information (Sheet 1 of 2)

Information Type	Example
Server details	• Machine number • Process number • Client number • Application name • Language details • User profile information • Cookies
Details about the connection process	• First connection, including server name, connect port, redirect. • First port's connection, including, initial socket, port, local port, server name, connect port, server application name, username, password, profile and shared variables. • First connection timestamp. • Second connection timestamp.
Details about CreateFrame messages	2/12/03 11:35:55 AM CreateFrame: hComp=21, libraryName PRN400X, lpszFormName = JITGUI, lpWindowName = , hCompParent= 1, X= -32768, Y = 0, nWidth = -32768, nHeight = 0
Keep alives sent	List of KeepAlive messages sent by the Client.
Keep alives received	List of KeepAlive messages received by the Client.

Table 36. Client dump: session level information (Sheet 2 of 2)

Information Type	Example
Outgoing requests	2/12/03 11:35:50 AM Client ->Server: Command hComp=0, id=22, action=8
Incoming requests	2/12/03 11:36:00 AM Server -> Client: createWindowControls
Sent shared user variables	2/12/03 11:35:50 AM Sent user variables:[ ]
Received shared user variables	A list of received shared user variables
User messages	2/12/03 11:35:41 AM Hello world
Chronological order of events	A list of all the session's events in chronological order.

Table 37. Client dump: exception information

Information Type	Example
A list of exceptions	java.lang.Exception: Quit Stack - Stack trace of the terminating thread at cst/debug/CoreDump.saveStackTrace at cst/common/general/ CoreDump.storeQuitReason at cst/client/comm/CommServer.run

Session Core Dump File

The Session Core Dump file includes the following details:

Table 38 lists the type of information found in the General Process information section.

Table 39 lists the type of information found in the Session Level information section.

Table 40 lists the type of information found in the Exception information section.

Table 38. Session dump: general process information (Sheet 1 of 3)

Information Type	Example
Java vendor and version number	Java vendor: Sun Microsystems Inc., version: 1.4.2_02
OS name and architecture number	OS name: Windows 2000, architecture: x86
JIS version or PTF name	Version/PTF Name = 8.1
JIS version Build number	BuildNumber = 8,1,0,54
Machine IP address	Machine IP address: 10.90.18.149
JVM memory consumption	Memory: Free: 49 Kb (3%) Total: 1488Kb
Session termination details	Session was terminated due to: 2/12/03 11:36:00 AM Quit reason: Exception in CommServer: java.lang.NullPointer Exception
Session details	• Session ID • Application name • Application directory • Connection settings

Table 38. Session dump: general process information (Sheet 2 of 3)

Information Type	Example
Recent Subapplications	2/12/03 12:03:26 PM SubApplication: Name = MAIN, With window = true, With screen = false, Library number = 0, Library name = PRN400X, To bleed through = false, Auto synchronize = false, Screen ID = null, Is master = false, Is dependent = false, Cursor place = cst.server.struct.GALPoint@8df60, Focused control name = , Is principal = false, Is dependent = false, Transition type = , Host screen name = MAIN, Is GDS = false, GDS type = 0, Next transaction = null, Next program = null, Auto synchronize = false
Emulator events	2/12/03 12:03:24 PM [Session 1 Telnet] Screen changed = false, Keyboard state changed = true, Keyboard locked = false, Cursor moved = false, MW state changed = false, MW = false, Screen size = 0
CreateFrame messages	2/12/03 12:03:22 PM CreateFrame: PRN400X, JITGUI,
Input XMLs	• Last EndUserAction message • Last SendWindowData message
Output XMLs	• Last EndUserAction message • Last SendWindowData message
Keep alives sent	A list of KeepAlive messages sent by the session.
Keep alives received	A list of KeepAlive messages received by the session.
Outgoing requests	2/12/03 12:03:32 PM Server -> Client: DestroyWindow
Incoming requests	2/12/03 12:03:18 PM Client -> Server: SendWindowData
Sent shared user variables	A list of sent shared user variables.

Table 38. Session dump: general process information (Sheet 3 of 3)

Information Type	Example
Received shared user variables	A list of received shared user variables.
Activated methods	1/13/04 11:53:36 AM u_SelectMenuOption lp: 0
User messages	A list of method comments.
Cycle number	Information about internal data structure.
Chronological order of events	A list of all the session's events in chronological order.
A list of exceptions	java.lang.Exception: Quit Stack - Stack trace of the terminating thread at cst.debug.CoreDump.save StackTrace(CoreDump.java:94) at cst.common.general.Core Dump.storeQuitReason(CoreDump.java:112) at cst.server.applicat.MainSub Application.window Destroyed(MainSubApplication.java:60)

Table 39. Session dump: session level information (Sheet 1 of 3)

Information Type	Example
Session details	• Session ID • Application name • Application directory • Connection settings

Table 39. Session dump: session level information (Sheet 2 of 3)

Information Type	Example
Recent Subapplications	2/12/03 12:03:26 PM SubApplication: Name = MAIN, With window = true, With screen = false, Library number = 0, Library name = PRN400X, To bleed through = false, Auto synchronize = false, Screen ID = null, Is master = false, Is dependent = false, Cursor place = cst.server.struct.GALPoint@8df60, Focused control name = , Is principal = false, Is dependent = false, Transition type = , Host screen name = MAIN, Is GDS = false, GDS type = 0, Next transaction = null, Next program = null, Auto synchronize = false
Emulator events	2/12/03 12:03:24 PM [Session 1 Telnet] Screen changed = false, Keyboard state changed = true, Keyboard locked = false, Cursor moved = false, MW state changed = false, MW = false, Screen size = 0
CreateFrame messages	2/12/03 12:03:22 PM CreateFrame: PRN400X, JITGUI
Input XMLs	• Last EndUserAction message • Last SendWindowData message
Output XMLs	• Last EndUserAction message • Last SendWindowData message
Keep alives sent	A list of KeepAlive messages sent by the session.
Keep alives received	A list of KeepAlive messages received by the session.
Outgoing requests	2/12/03 12:03:32 PM Server -> Client: DestroyWindow
Incoming requests	2/12/03 12:03:18 PM Client -> Server: SendWindowData
Sent shared user variables	A list of sent shared user variables.

Table 39. Session dump: session level information (Sheet 3 of 3)

Information Type	Example
Received shared user variables	A list of received shared user variables.
Activated methods	1/13/04 11:53:36 AM u_SelectMenuOption lp: 0
User messages	A list of method comments.
Cycle number	Information about internal data structure.
Chronological order of events	A list of all the session's events in chronological order.

Table 40. Session dump: exception information

Information Type	Example
A list of exceptions	java.lang.Exception: Quit Stack - Stack trace of the terminating thread at cst.debug.CoreDump.save StackTrace(CoreDump.java:94) at cst.common.general.Core Dump.storeQuitReason(CoreDump.java:112) at cst.server.applicat.MainSubApplication.window Destroyed(MainSubApplication.java:60)

Adding Messages

In addition to the messages provided by webMethods JIS, you can add your own messages to the session dump file. This is done using the function:

`public void storeUserMessage(String message)` defined in the `cst.common.general.ICoreDump` interface.

- To retrieve the implementation of this interface from a client extension code: Use the function `getClientDump()` from the JacadaStarter.java file.

  For example, add a function:
  `getClientDump().storeUserMessage(“message”);` in
  `<ApplName>\user\JacadaStarter.java, method init()`
- To retrieve the implementation of this interface from a server extension code: Use the function `getSessionDump()` from the Global.java file.

  For example, add a function:
  `globals_parm.getSessionDump().storeUserMessage (“message”);`
  in the constructor of `<ApplName>\user\TheApplication.java`

Checking Server Configuration

Checking the configuration of the server is an important stage in the development of the application. By using the Server Configuration Checker you can check that your server configuration is free of inaccuracies, before deploying the application. Certain inaccuracies in the jacadasv.ini file prevent the server from running, whereas others can cause unexpected behavior.

There are two types of server inaccuracies:

- Local inaccuracies, where single properties are not defined according to their legal properties, such as missing property definitions, non-numeric values for numeric properties, and illegal characters in property definitions.
- System inaccuracies, where combinations of property definitions are not defined correctly. For example, if the port range does not fit the number of processes.

Most local inaccuracies are reported in the debug logs. However, some local mistakes are not reported, such as non-reasonable values. For example, if a parameter that should be defined in milliseconds is defined in seconds, this is considered a non-reasonable value.

System inaccuracies are more elusive due to the decentralized nature of the JIS Server. Each server process reads the jacadasv.ini on its own machine, but only extracts definitions that relate to the placement of the process in the scalability tree. There is no central location responsible for validating the entire server configuration.

Server Configuration Checker

The Server Configuration Checker analyzes the jacadasv.ini file and identifies local and system-level inaccuracies at all scalability levels. The checker looks for and reports configuration errors (see Table 41) and configuration warnings (see Table 42) at different debug levels.

In addition to running in Server Mode in every server process, the Server Configuration Checker can also be run in Offline Mode.

Table 41. Server configuration errors

Configuration Element	Possible Error
ServerMachines Section	Section is not defined. Section does not define a machine.
Server Port Range	Range definition is invalid. Not all server processes have two ports.
http and https Port Range	Range definitions are invalid. Not all server processes have one http and one https port. The http test is only executed if `HttpClient = TRUE`. The https test is only executed if `HttpsClient = TRUE`.
Registry/RMI Port Range	Port range is invalid. Not all processes have one RMI registry and one node registry port.
Session Handling	None of the processes can handle sessions.
jacadasv.ini File	Exceptions occur when reading and analyzing the `jacadasv.ini` file.

Table 42. Server configuration warnings

Configuration Element	Possible Warning
http/https Port Range	http/https Level1 port range contains more than one port.
Session Handling	A process does not handle sessions. This warning is not issued for Level1.
Property Values	A property value lies outside the allowed range. See “Range of Valid Properties” on page 181 for a list of valid ranges.
Unrecognized Properties	The server does not recognize known properties. The warning acts as protection against spelling mistakes.
maxProcessSessions	The maxMachineSessions value is smaller than the sum of maxProcessSessions values for the required processes.
Numeric Properties	Numeric properties have non-numeric values.
Negative Values	Properties have negative values.

Enabling the Server Configuration Checker

Server Mode

In Server Mode, the Server Configuration Checker tests the configuration of the machine on which it is run. By default, the main process of every server machine executes the checker upon starting. If the checker identifies an error, the server aborts. If the checker identifies a warning, information is written to a log, but the checker does not prevent the server from starting.

In Server Mode, the Server Configuration Checker reports errors and warnings at the following debug levels:

- Errors are reported at debug level 0.

- Warnings are reported at debug level 1.
- Normal report messages are reported in debug level 50.

In Server Mode, the Server Configuration Checker writes the configuration information to:

Log Name	`debug_1.txt`
Log Location	`<InstallDir>\JacadaFiles\classes\logs`

To disable the Server Configuration Checker:

In the `jacadasv.ini` file, in the `[GeneralParameters]` section, set the `checkServerConfiguration` parameter to 0.

Offline Mode

You can also run the Server Configuration Checker offline. When run offline, the checker reads the jacadasv.ini file and analyses the configuration of all the defined server machines. The checker tests the configuration and then exits, without starting the server.

In Offline Mode, the Server Configuration Checker writes the configuration information to:

Log Name	`debug_start.txt`
Log Location	`<InstallDir>\JacadaFiles\classes\logs`

To enable the Server Configuration Checker in Offline Mode:

In the `jacadasv.bat` file, add the `-c` command line option.

See “The JIS Server Command Line Parameters” on page 103 for information about the `jacadasv.bat` command line options.

Range of Valid Properties

One of the tasks that the Server Configuration Checker performs is to identify non-reasonable property values. Following is a list of the valid ranges of the numeric properties in the jacadasv.ini file:

Table 43. Jacadasv.ini: valid numeric parameters (Sheet 1 of 3)

Property Name	Description	Default Value (seconds)	Valid Range
KeepAliveTimerTick	Time interval for sending keep alive to parent process.	60	10-600
KeepAliveTimeout	Process inactivity timeout.	10	1-100
SystemConnection TimeOut	Time interval for establishing connection between machines.	120	20-1200
RegistrySpawn TimeOut	Timeout for spawning the registry.	45	10-450
WaitForSpawned	Time interval for waiting for a spawned process.	60	20-600
WaitForStatus	Time interval for session allocation.	100ms	10-1000
AllocLockTimeout	Timeout for process lock during session allocation.	10	1-600
ExpectingSession Timeout	Timeout for client redirection.	300	10-3000
SessionIdleTimeout	Idle timeout for XML and JCS sessions.	180* 3600**	0-86400
RMISocketTimeout	Timeout for RMI calls.	20	10-200
MsgboxTimeout	Timeout for message box reply.	36000	3600-360000

Table 43. Jacadasv.ini: valid numeric parameters (Sheet 2 of 3)

Property Name	Description	Default Value (seconds)	Valid Range
PanelTimeout	Timeout for next panel selection.	36000	3600-360000
GetTextFromUser Timeout	Timeout for getting user text.	36000	3600-360000
KeepAlive	Client keep alive.	240	0-2400
RecvTimeout	Timeout for client reply.	200	60-2000
ResponseTimeout	Http response timeout.	100	20-1000
SessionInitialization TimeoutInSeconds	Timeout for preloaded session initialization.	300	10-3000
MaxNewSessions PerSecond	Maximum number of pool sessions that can be created per second.	5	1-30
PoolCreationDelay InSeconds	Number of seconds that the server waits before starting to fill the pools.	30	1-1800
InitialPoolSize	Initial size of the pool.	10	0-1000
OnGoingPoolSize	Ongoing size of the pool.	10	0-100
ValidityCheck IntervalInSeconds	Frequency of the validity checks.	300	10-3600
SizeCheckInterval InSeconds	Frequency of the checks whether the pool should be refilled.	10	1-3600
StartUpSessions Percent	Percent of sessions supported in server start.	0	0-100

Table 43. Jacadasv.ini: valid numeric parameters (Sheet 3 of 3)

Property Name	Description	Default Value (seconds)	Valid Range
SpareSessions Percent	Percent of spare sessions in ongoing state.	0	0-100

* This is the default value as defined in the `ServerConfiguration.xml` file for the JIS Server, and in the `session-timeout` parameter in the `web.xml` file when using a J2EE application server.

** This is the default value when using the JIS Server if no value is defined in the `ServerConfiguration.xml` file nor in the [HTTP] section of the `jacadasv.ini` file.

JIS Administrator

There are two versions of the JIS Administrator: one for use with the proprietary JIS Server, and one for use with a third-party application server or Web container.

Note: JIS 9.0.4 introduces a 3rd option for running the J2EE version of Jam using the proprietary server. For more information see the JIS 9.0.4 release notes.

This section about the JIS Administrator is divided into three main subsections:

- JIS Administrator for the webMethods JIS Proprietary Server
- JIS Administrator for the J2EE Environment
- JIS Administrator Interfaces Common to Both Environments

JIS Administrator for the JIS Proprietary Server

The JIS Administrator **(JAM)** provides the system administrator with a means of viewing and controlling the JIS Server

You can view the structure of the JIS Server, and the process and session activity taking place on the server system.

You can control the JIS Server's operation as follows:

- Pause and resume its activity.
- Stop its execution altogether.
- Change selected JIS Server configuration settings.
- Change JIS Server debug settings for the current run or permanently.

Starting the JIS Administrator

You can start the JIS Administrator from either the Server Machine, where the server machine is running Windows, or remotely from a Windows workstation mapped to the Server Machine. Use the latter case to run the JIS Administrator from a remote site, or when the server is not installed on Windows.

Starting the JIS Administrator from the Server Machine

In the **Startup** menu, select **Programs > JIS > JIS Administrator**.

A wait cursor is displayed until the JIS Administrator interface opens on the screen.

Note: If the JIS Server is not running, the interface will open without displaying any contents.If the server is running then when the interface opens, it displays the server's status.

running Windows

Note: Make sure **-p http://localhost:8080/port** is present in the jam.bat command line. You can replace //localhost:8080/ with the IP address of the server on which jam.bat is located.

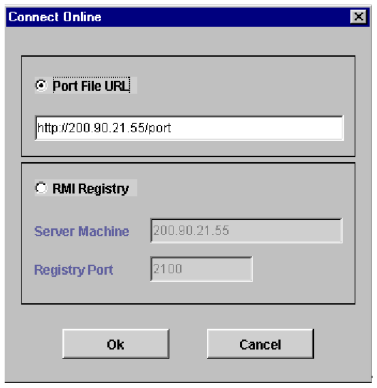
Connecting Online to the JIS Server

To connect online to the JIS Server you either need to supply the complete URL for the server machine, or the server machine's IP address. This information is entered in the **Connect Online** dialog box.

To open the **Connect Online** dialog box:

1 Either click the button on the tool bar or select **Connect** from the **Connect** menu.

The **Connect Online** dialog box opens:



Port File URL: http://200.90.21.55/port
Server Machine: 200.90.21.55
Registry Port: 2100

Figure 37. Connect online dialog box

2 Choose either:
- The **Port File URL** option.

-OR-
- The **RMI Registry** option.

3 Enter the appropriate information as indicated in Table 44.

Table 44. Connect Online parameters

Parameter	Description
Port File URL	Enter the complete URL as indicated in the illustration. Make sure to include the: • IP address and port of the JIS Server. • followed by **/port**.
RMI Registry	When using the RMI registry you must supply the: • Server Machines IP address. • Registry Port as indicated in the `jacadasv.ini` file.
Server Machine	Enter the server machine's IP address in the text box.
Registry Port	Enter the default value unless it was changed in the `jacadasv.ini` file. In that case, enter the value recorded in the `jacadasv.ini` file. The default registry port is **2100**.

Debugging the JIS Administrator

The JIS Administrator debugging feature enables you to keep track of the activity registered in the JIS Administrator.

The debug information is written to a log file called `debug_JAM.log`. This file is created automatically once the debugging feature is activated. The file is placed under a designated directory.

To activate the JAM debugging feature:

Add any of the following switches in Table 45 to the JIS Administrator parameter line:

Table 45. JAM debugging feature: activating parameters

Switch	Values and Description
-d	Debug level. *Example*: `-d 50`
-l	Debug logging file directory. *Example*: `-l c:\temp`
-h	Optional parameter that prompts the JIS Administrator console. The console includes the syntax of the JIS Administrator startup command and a list of other command line options.

Example 4. JAM startup command

The following is an example of the JIS Administrator startup command:

```
JAM.bat -d 50 -l c:\temp
```

Do not forget to leave a space between the parameter and its value.

JIS Administrator for the J2EE Environment

The JIS Administrator provides the system administrator with a means of viewing the process and session activity taking place on the a given server or cluster of servers. Specific session and application properties can also be modified through the JIS Administrator.

Running the JIS Administrator Under J2EE

To run the JIS Administrator Runtime Configuration interface, open a browser and go to the URL
`http://<AppServer IP addr>:<port number>/JacadaAdmin/admin`

Be sure to use the correct port number of the application server (administrative or stand-alone) to which the `JacadaAdminApplication.ear` was deployed.

The JIS Administrator Interfaces

The JIS Administrator tool is divided into two interfaces:

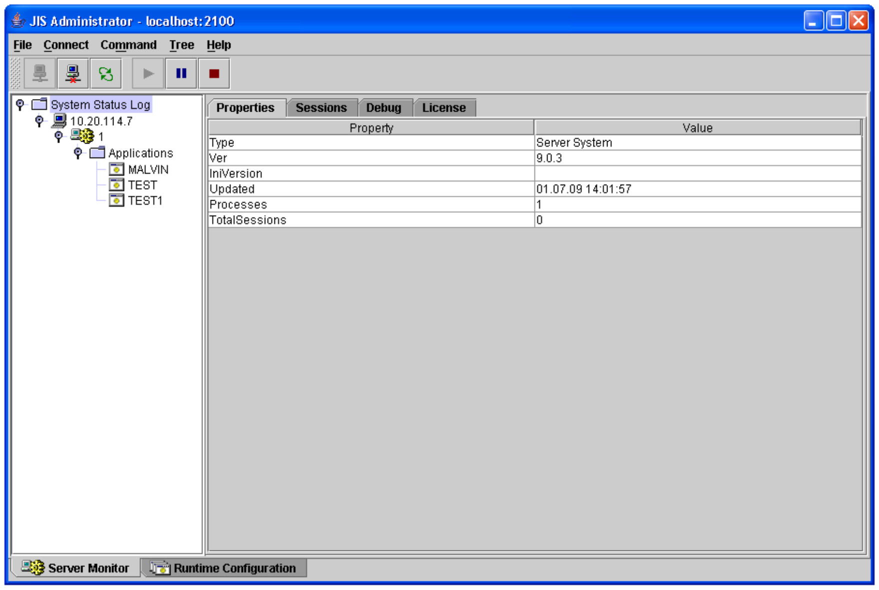
- The **Server Monitor** interface
- The **Runtime Configuration** interface

To move between interfaces, use the appropriate tabs on the bottom left corner of the tool.

The Server Monitor Interface

The Server monitor interface is divided into five main regions:

- The **System Status Log** pane.
- The **Properties** tab.
- The **Sessions** tab.
- The **Debug** tab.
- The **License** tab.



Property	Value
Type	Server System
Ver	9.0.3
IniVersion	
Updated	01.07.09 14:01:57
Processes	1
TotalSessions	0

Figure 38. The system status log pane, with the Properties tab selected

The System Status Log Pane

The **System Status Log** pane displays a hierarchical representation of the JIS Server system structure. The **System Status Log** pane displays the hierarchy in a tree structure composed of three levels: Server Machines resident on the System, Processes running on any particular Server Machine, and Applications and Sessions active on any process.

You can expand and collapse the System Status Log to display any particular level by double clicking the system element that interests you.

The or symbol appears immediately before each interface element.

The symbol indicates that the system element can be further expanded to display its sub elements, whereas the symbol indicates that the system element is expanded and is displaying its sub elements.

The Properties Tab

The **Properties** tab displays Server System information relevant to the particular level or system element highlighted in the System Status Log Pane. Information is categorized in the form of Property and Value of each property. Table 46 describes the fields in the Properties tab.

The Sessions Tab

The **Session** tab lists each and every session that is active on the server system and displays the values for the properties in a tabular format. The column order can be arranged to view the information that is important at the time, and the columns can be sorted alphanumerically from top to bottom or from bottom to top.

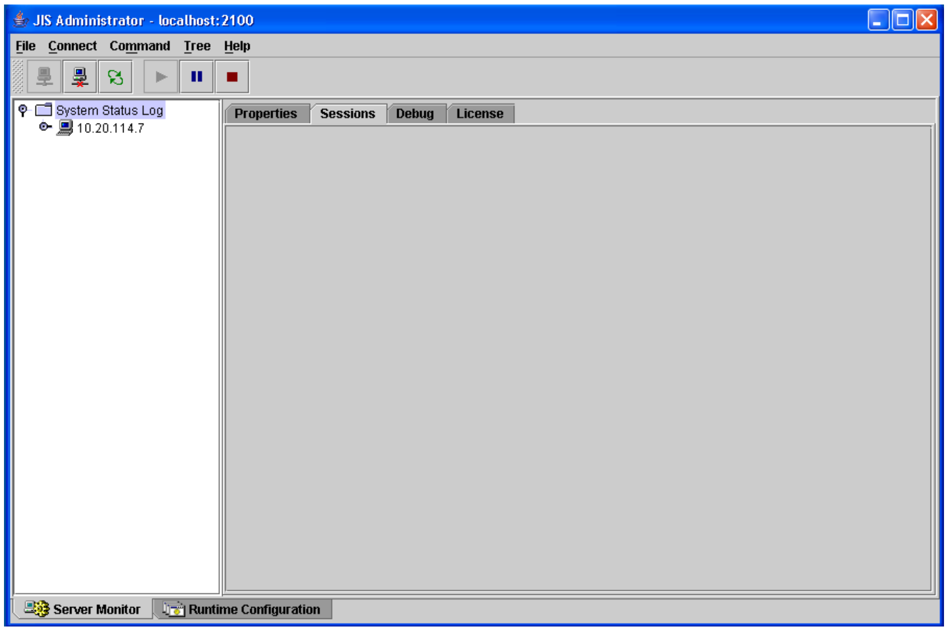


Figure 39. The Sessions tab

The fields displayed in the Session tab include:

- Server
- ID
- Created
- User
- UserAddress
- ApplicationName
- LibraryName
- CurrentScreen
- State
- Transactions
- TotalDuration
- AvgDuration
- LastTransaction
- LastTransactionDuration

See the section for the Specific Session element in Table 46 for a description of these fields.

The Properties and Sessions Tabs

The properties and values listed in the **Properties** tab vary according to the element in focus in the **System Status Log** pane. The properties in the Properties and Sessions tabs are described in Table 46.

Table 46. Properties in the Properties and Sessions tabs(Sheet 1 of 4)

System Status Log Pane Element	Property Name	Value Description
System Status Log		
	Type	JIS Server System.
	Ver	webMethods JIS version release number.
	IniVersion	Lists the IniVersion, if specified in the INI file.
	Updated	Last time that the information in the display was refreshed.
	Processes	Total number of processes running on the system.
	TotalSessions	Total number of sessions running on the system.
Server Machine		
	Address	Server Machine IP Address.
	RMI Port	RMI port number.
	Processes	Total number of processes running on the server machine.

Table 46. Properties in the Properties and Sessions tabs(Sheet 2 of 4)

System Status Log Pane Element	Property Name	Value Description
	TotalSessions	Total number of sessions running on the server machine.
Process		
	Alias	A number describing the hierarchical position of the process within the system.
	Port1	Port Number the process uses.
	HttpPort	The HTTP port number.
	HttpsPort	The HTTPS port number.
	State	Indicates the activity state of the process. Possible values: INITIALIZING, STARTED, PAUSED, STOPPED, PENDING_START, PENDING_STOP, FAILED
Application		
	Applications	Number of Applications running on the process.
<ApplName>		
	Name	Application name.
	Sessions	Number of sessions on the process that are being used by the Application.

Table 46. Properties in the Properties and Sessions tabs(Sheet 3 of 4)

System Status Log Pane Element	Property Name	Value Description
	TotalPenalty	Total penalty ranking experienced by the process.
Sessions Folder		
	Spare	The number of free sessions available for use on the process.
	Size	The number of sessions in use on the process.
Specific Session		
	Server	Server the session is running on.
	ID	Unique numeric Session ID, since the server was started.
	Created	Date and time session initiated.
	User	User profile.
	User address	IP address of the client machine.
	ApplicationName	Name of Application being run by the session.
	LibraryName	Name of library being used by the session, if any.
	CurrentScreen	Screen name currently active.
	State	Event occurring on screen. See “JIS Server Logging Support” on page 159 for details.

Table 46. Properties in the Properties and Sessions tabs(Sheet 4 of 4)

System Status Log Pane Element	Property Name	Value Description
	Transactions	Total number of host application transactions since initiating current session.
	TotalDuration	Total server processing time for transactions in milliseconds.
	AvgDuration	Average duration of each transaction in milliseconds.
	LastTransaction	Start time of last transaction.
	LastTransactionDuration	Duration of the last transaction in milliseconds.

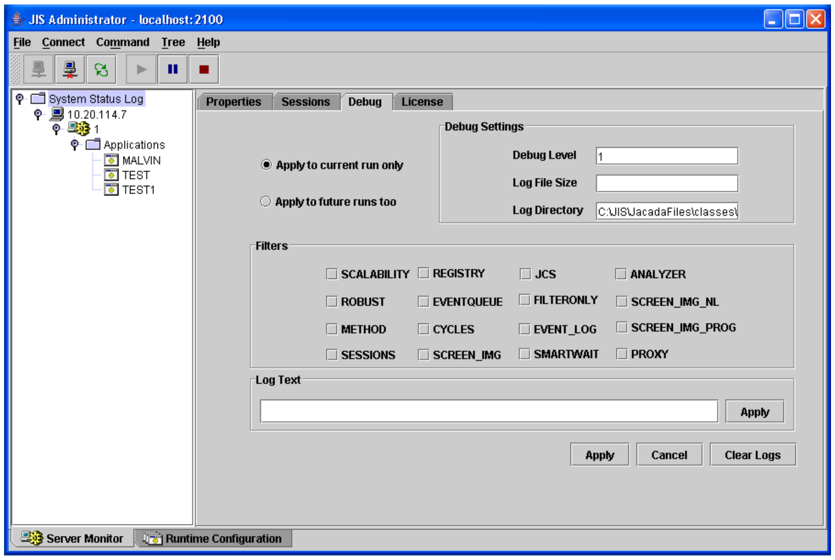
The Debug Tab

The Debug tab displays settings of the `jacadasv.ini` file that are related to debugging. When the tab is opened, it displays the current values of the settings. The values of the displayed debug settings can be changed, for the duration of the current server run, or permanently, if so desired. If you choose to make the changes permanent, the `jacadasv.ini` file is updated with the new values.

The facilities available on the Debug Tab are:

- The Debug level can be changed.
- The Debug Log File size can be changed.
- The Debug Log directory can be changed.
- Log Filters can be turned on and off.
- A text string of your choice can be written to the log as a marker.

An example of the contents of the Debug tab is shown in Figure 40. The fields in the Debug tab are described in Table 47 on page 196.



[✓ Apply to current run only] Apply to future runs too

Debug Level: 1

Log File Size:

Log Directory: C:\JIS\JacadaFiles\classes\

Figure 40. The Debug tab

Table 47. Elements in the JIS Administrator Debug tab (Sheet 1 of 2)

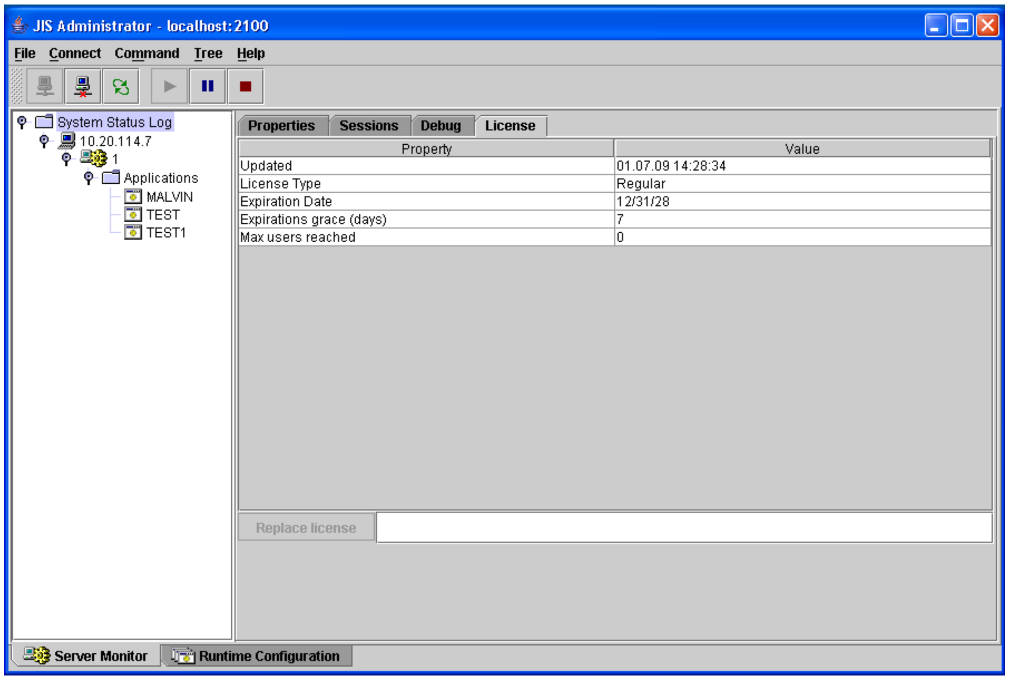
Debug Tab Element	Description
Apply to current run only	When selected, the changes you specify in the debug tab of the JIS Administrator are effective only for the duration of the current run of the Server.
Apply to future runs, too	When selected, the changes you specify are effective for the current run and for all future runs until changed. The new values you specify are written to the `jacadasv.ini` file.
Debug Settings	
Debug Level	Corresponds to the `RtDebugLevel` INI file setting, described in “RtDebugLevel” on page 111.
Log File Size	Corresponds to the `RtDebugFileMaxSize` INI file setting, described in “RtDebugFileMaxSize” on page 110.
Log Directory	Corresponds to the `RtLogsDir` INI file setting, described in“RtLogsDir” on page 112.

Table 47. Elements in the JIS Administrator Debug tab (Sheet 2 of 2)

Debug Tab Element	Description
Filters	Log filters give you the ability to selectively print specific classes of log messages to the debug log. Instead of setting a high debug level and examining the entire debug log for messages related to a particular issue, you would set the debug level to 1 simply to turn on debugging, and utilize a log filter or filters to produce only those messages related to the issue in question. You normally use log filters in the process of investigating a particular application problem or performance issue. The messages produced by many of the debug filters are rather obscure and of limited use to the customer unassisted. There are a few debug filters, though, that can be of practical use to the customer unassisted; see “Debug Filters” on page 166.
Log Text	Lets you specify any character string to be written to the log file as a marker. For example, you may want a marker in the logfile to signal the point in time where you modified the debug settings.
Apply	Writes the character string to the log file.
Other buttons	
Apply	Applies the modifications to the server.
Cancel	Nullifies any changes applied to the debug settings on the screen since the last time the “Apply” button was clicked.
Clear Logs	Use the Clear Logs button to reset the logging point to the beginning of the log file.
Help	Invokes JIS Administrator online help.

The License Tab

The License tab of the JIS Administrator is used for displaying the details of the runtime license, and for updating the runtime license when necessary. An example of the contents of the License tab is shown in Figure 41. The fields in the License tab are described in Table 48.



Property	Value
Updated	01.07.09 14:28:34
License Type	Regular
Expiration Date	12/31/28
Expirations grace (days)	7
Max users reached	0

Figure 41. The License tab

Table 48. Elements in the JIS Administrator Debug tab

License Tab Element	Description
Updated	The date the product license was last updated.
License Type	Possible license types are: • Regular - specifies a maximum permitted number of simultaneous user sessions, and expires on the date shown. • Date Limited - license expires on the date shown; no limit on number of simultaneous users. • Users Limited - no expiration date, but sets a maximum permitted number of simultaneous users. • Unlimited
Expiration date	Date license expires. Contact your Software AG representative at least two to four weeks before the expiration date to arrange for a license update.
Expiration grace	Additional days after the formal expiration date during which the current license will continue to be valid.
Max users reached	Greatest number of simultaneous users seen by the Server during the current Server execution.
Replace license	This is a button, for use when the time comes to update your license. To update the license: paste or type the new license key in the text box to the right of the Replace license button and click on the button.

Operations you Perform Using the Server Monitor

The most typical use of the Server Monitor is to manipulate the JIS Server and to view the distribution of Server machines, Applications, processes and sessions on the server system. Under certain circumstances your needs may require you to:

- Pause/Resume/Stop the JIS Server.
- Save/Open the System Status Log to file.
- Close sessions.

You can use the JIS Administrator to **Pause**, **Resume** or **Stop** the JIS Server. Each of these can be performed on either a single machine or on the entire farm.

You can either:

- Click the **Machine** icon 194.90.17.80 and choose the action you wish to perform on a single machine.

  -OR-
- Click the **SystemStatusLog** icon SystemStatusLog and choose the action you wish to perform on the entire server farm.

Note: When you pause or stop a single machine, the JIS Server remains active for other machines in the server farm.

Pausing the JIS Server

The Pause action changes the Server's status from Started to Paused. Once the status of the Server has changed, it will no longer allow new sessions to connect.

To pause the JIS Server:

1 Stand on the **Machine** icon or on the **SystemStatusLog** icon.

2 Click the button on the tool bar or select **Pause** from the **Command** menu.

Note: Once the status of the Server changes to Paused, the **Pause** button is disabled and the **Resume** button is enabled.

Resuming the JIS Server

The Resume action changes the Server's status from **Paused** to **Started**. Once the status of the Server has changed, it allows new sessions to connect.

To resume the JIS Server:

1 Stand on the **Machine** icon or on the **SystemStatusLog** icon.
2 Click the ▶ button on the tool bar or select **Resume** from the **Command** menu.

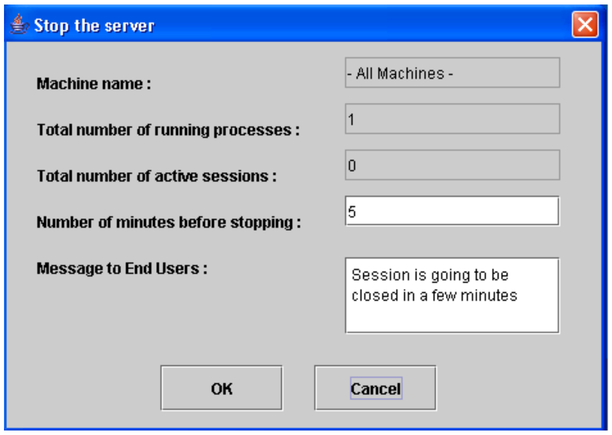
Note: Once the status of the Server changes to Started, the **Resume** button is disabled and the **Pause** button is enabled.

Stopping the JIS Server

The Stop action shuts down the JIS Server, terminating thereby all of its processes. Before shutting down, the JIS Server enters a `Pending_Stop` stage, during which it behaves as if it were paused. The time the JIS Server remains in `Pending_Stop` status depends on the setting defined in the **Stop JIS Server** dialog box. This is seen in the image below.

To stop the JIS Server:

1 Stand on the **Machine** icon or on the **SystemStatusLog** icon.
2 Click the ■ button on the tool bar or select **Stop** from the **Command** menu.
The **Stop JIS Server** dialog box opens:



Machine name : - All Machines -
Total number of running processes : 1
Total number of active sessions : 0
Number of minutes before stopping : 5
Message to End Users : Session is going to be closed in a few minutes

Figure 42. Stop JIS Server dialog box

3 Fill in the fields according to the instructions in Table 49.

Table 49. Stop JIS Server instructions

Field	Description
Machine name	Displays the name of the machine that will be closed, or "All Machines", if you have chosen to shut down the server.
Total number of running processes	Displays the number of processes currently running.
Total number of active sessions	Displays the number of sessions currently active.
Number of minutes before stopping	Determines the number of minutes the server will be in `Pending_Stop` status before stopping. This field can be edited.
Message to End Users	Defines the message to be displayed to the end user when stopping the JIS Server. This field can be edited.

4 Click **OK**.

Saving the System Status Log to File

The System Status Log display is periodically updated with new information. The interval between updates is set in the `jacadasv.ini` file. By default, the update interval is set to 30 seconds. Background information can be found in the sections on "Scalability" on page 139, and "JIS Server Logging Support" on page 159.

For any number of reasons you may need to save an instance of the System Status Log display in order to review the state of the server system. JIS Administrator saves the System Status Log in XML format.

To save the System Status Log to file:

1 From the **File** menu, select **Save**.
  The **Save XML File** dialog box opens.
2 Select a directory to store the log file.
3 Enter a name for the log file in the **File Name** edit box.
4 Click **Save**.

Opening a Saved System Status Log

To open a saved system status log file:

1. From the **File** menu, select **Open**.
   The **Open XML File** dialog box opens.
2. Browse for the directory that houses the saved log file, and select the file.
3. Click **Open**.

The log file opens and the System Status Log is displayed in JIS Administrator.

Closing Sessions

You can either close a specific session or close all the sessions of a specific server process.

To close a specific session:

1. Select the specific session node in the **System Status Log** pane.
2. From the **Command** menu, select **Close Session** or press the keyboard **Delete** key.
3. A confirmation message appears with the message:
   ```
   Do you want to close session <sessionID> in process
   <processAlias> machine <machineName>?
   ```
4. Click **Yes**. The session closes.

This option is only enabled if a session is selected in the **System Status Log** pane.

To close all sessions of a server process:

1. Select the process node in the **System Status Log** pane.
2. From the **Command** menu, select **Close Process Sessions** or press the keyboard **Delete** button.
3. A confirmation message appears with the message:
   ```
   Do you want to close all the sessions in process
   <processAlias> machine <machineName>?
   ```
4. Click **Yes**. The sessions close.

This option is only enabled if a process is selected in the **System Status Log** pane.

Viewing All Columns on the Session Tab

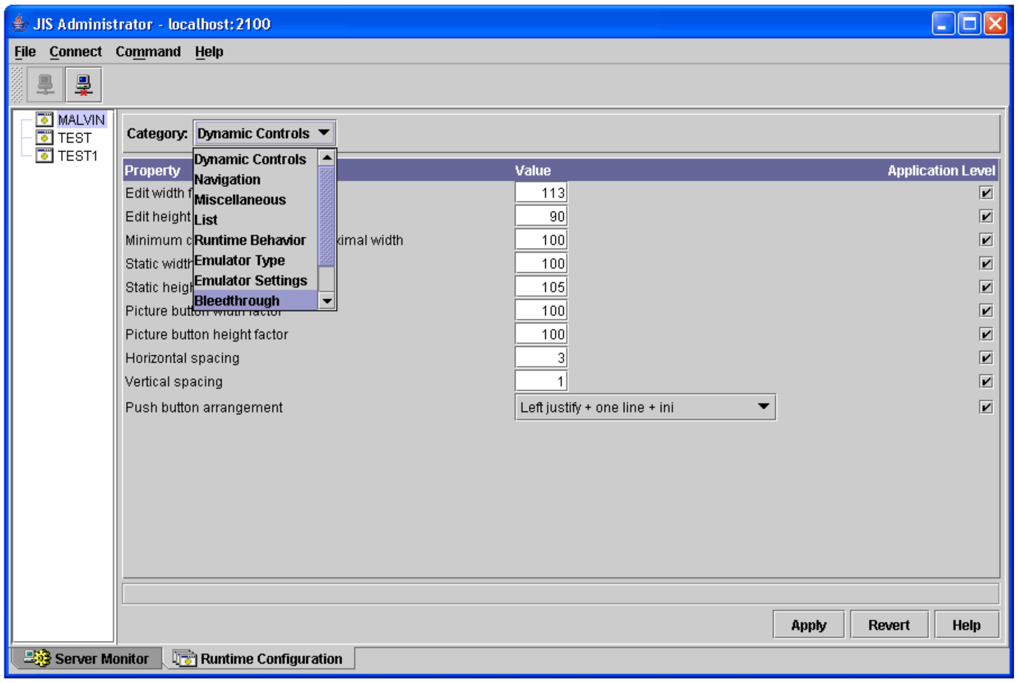
Not all columns in the **Sessions** tab are always in view.

To view a particular column you may have to use the horizontal scroll bar to bring a particular column into view.

The Runtime Configuration Interface

This section applies to JIS Administrator on the webMethods JIS Proprietary Server, and also to JIS Administrator running on a third-party (J2EE) application server.

The behavior of the runtime environment is independent of any specific application. Runtime behavior can be reset each time the runtime environment is entered—without touching the application executable. When you run an application for the first time, the runtime environment is created with certain default values. You can, if you wish, change these values. The new values are then automatically preserved between runtime sessions.



Category: Dynamic Controls

Property	Value	Application Level
Edit width f	113	✓
Edit height	90	✓
Minimum c ximal width	100	✓
Static width	100	✓
Static heigl	105	✓
Picture button width factor	100	✓
Picture button height factor	100	✓
Horizontal spacing	3	✓
Vertical spacing	1	✓
Push button arrangement	Left justify + one line + ini	✓

Figure 43. The runtime configuration interface

The Runtime Configuration interface provides you with an option to implement these changes:

Table 50. Runtime configuration interface components (Sheet 1 of 2)

Component	Description
Application Tree	In the left pane, displays a list of Applications, libraries, and user profiles you can run on the JIS Server. Highlight the desired component whose runtime values you wish to view or change.

Table 50. Runtime configuration interface components (Sheet 2 of 2)

Component	Description
Category	The runtime options are grouped into logical categories. The categories are listed in the **Category** combo box.
Property	Lists all the parameter names included within a specific category.
Value	Insert parameter values, or change existing ones.
Level	Depending upon whether you are running an Application or a User Profile this column's heading reads Application Level or User Profile Level respectively. Accordingly, setting a parameter's check box determines the level on which the changes take place. For more information, see "Application Level vs. User Profile Level" on page 205.
Apply	Automatically writes the changes you have made to this session of the Application to the runtime ini file.
Revert	Clears ALL the changes you have made to this session of the Application.

Application Level vs. User Profile Level

Changes to values can be performed on two levels:

- An **Application** level
  -OR-
- A **User Profile** level.

Changes on an Application level affect all users of the specific Application or library; changes on a User Profile level affect only that user whose profile is being used.

Hierarchically, User Profile level settings have precedence over Application level settings, which, in turn, have precedence over the hard-coded default settings provided by webMethods JIS.

To change parameter values on an Application or Library level:

1 In the Application Tree, in the left hand pane, select the Application whose values you wish to change.

2 In the Application Level column, check the parameter you wish to change.

  The following message appears:

  “`Do you want to explicitly set the current value for this Application Level?`”

3 Click **Yes**.

4 In the Value column, set the value you wish to change.

Note: The applied changes take effect in the next session.

To change parameter values on a User Profile Level:

1 In the Application Tree, in the left hand pane, select the Application and then the Profile whose values you wish to change.

2 In the User Profile Level column, check the parameter you wish to change.

  The following message appears:

  “Do you want to explicitly set the current value for this User Profile Level?”

  Click **Yes**.

3 In the Value column, set the value you wish to change.

Note: The applied changes take effect in the next session.

The Runtime Configuration Categories

The runtime configuration categories are as follows:

- Dynamic Controls
- Navigation
- Miscellaneous
- List
- Runtime Behavior
- Emulator Type
- Emulator Settings
- Bleedthrough
- Display
- XHTML

The parameters in each category are described in the following tables.

Dynamic Controls

Table 51. Runtime configuration: dynamic controls parameters

Property	Description
Edit box width factor	Represents a percentage relative to the original width of the Edit box control.
Edit box height factor	Represents a percentage relative to the original height of the Edit box control.
Minimum characters in Edit box for maximum width	Sets the width to be used when calculating the size of the Edit box control.
Static width factor	Represents a percentage relative to the original width of the Static control.
Static height factor	Represents a percentage relative to the original height of the Static control.
Picture button width factor	Represents a percentage relative to the original width of the Picture button control.
Picture button height factor	Represents a percentage relative to the original height of the Picture button control.
Horizontal spacing	Sets the amount of horizontal space, in pixels, between controls.
Vertical spacing	Sets the amount of vertical space, in pixels, between controls.
Push button arrangement	Push button arrangement for one dynamic group. The first puts all buttons in one line while the INI setting `DynamicControlsOrder` in the `[SubAppl]` name group can define a preferred order. The others leave the order as on the screen.

Navigation

Table 52. Runtime configuration: navigation parameters

Property	Description
Application libraries	List of libraries to be used during the current session. The list also includes the Application's name.
Maximum milliseconds for input inhibited	The maximum amount of time (in milliseconds) in which the hourglass is displayed.
Polling time	Frequency (in milliseconds) of polling the emulator connection.

Miscellaneous

Table 53. Runtime configuration: miscellaneous parameters (Sheet 1 of 2)

Property	Description
Load public formats	Displays common values from the INI file.
Multiple session support	Option to support multiple sessions.
Session title prefix	Sets the prefix of the main window title for each session.
Prefix character	Character that represents the *Windows* prefix character for accelerators.
Display message box on exit application	Displays the warning message before exiting an Application.
Application full name	Overwrites the short name given to the Application in ACE.
AutoStart	Skips the Application's **Startup** menu (the **Run** option).

Table 53. Runtime configuration: miscellaneous parameters (Sheet 2 of 2)

Property	Description
AutoSaveWindows Placement	When the user changes the size and position of a specific Subapplication in runtime, the new size and position are written to the application ini file. When the Subapplication is re-entered, its size and position are restored the size and position set by the user.

List

Table 54. Runtime configuration: list parameters (Sheet 1 of 2)

Property	Description
Save new column order	Saves column order after the user has dragged a column to a new position.
Tab stop on editable fields only	Tab stops only on editable fields only. Use the keyboard arrow keys to move into protected fields.
Tabbing from first/ last cell exits in table	Pressing the Tab keys on the first or last cell of a table exits the table.
Allow multi-page table	Displays as many host pages as fit into the table.
Multi-page multiplier value	The number of list pages read from the host and displayed in the table is a multiple of this value. For example, if the table displays two pages and the multiplier is three, then six pages are read from the host.
Check two consecutive pages for identical content	When set, two identical pages are recognized as the end of the list.
Move cursor before paging	Moves the host cursor to the list before paging.

Table 54. Runtime configuration: list parameters (Sheet 2 of 2)

Property	Description
Multi-select rows on RMB	Allows multiple selection of rows with a right mouse button click.
Host command characters for unselected records	A list of characters that are treated as an empty list command.

Runtime Behavior

Table 55. Runtime configuration: runtime behavior parameters (Sheet 1 of 2)

Property	Description
Protected combo box handling	Select the default handling for empty protected combo boxes. Choose between: • Disable • Disable or Hide if blank • Hide when blank is not a valid screen value • As on host
Hide protected check box with no text	Hides a check box with no header and no text when protected on the host.
Protected Edit box handling	When an Edit box field with text is protected on the host it can be disabled or made read-only with colors defined in the ini file. Choose between: • Disable • Disable or Hide if blank • ReadOnly with colors or Hide if blank • ReadOnly with colors • As on host
Ignore leading blanks in format	Ignores leading blanks when formatting values.

Table 55. Runtime configuration: runtime behavior parameters (Sheet 2 of 2)

Property	Description
Do not skip Subapplications	Used for debugging purposes only. When this option is checked, the runtime displays a window for all Subapplications that were set as either 'Never Display Window' or 'Conditionally Display Window'.
Refresh window when updated by host	When set, the runtime refreshes the window each time it is updated by the host. When cleared, any change to the window is ignored.
Date Base Year	The first year in a 100 year interval represented by a two-digit year format. For example, if 1949 is the base year, then 50 represents the year 1950 whereas 48 represents the year 2048.

Emulator Type

Table 56. Runtime configuration: emulator type parameters (Sheet 1 of 3)

Property	Description
Emulator type	Choose the type of emulator you wish to work with: • File • TN3270 • TN5250
File Emulator	
First panel to display	The first screen to be displayed out of the pool of panels. The advance mechanism of the panels is determined in the `panels.ini` file.
Internal TN3270	
Host address	Contains the name or the IP address of the host mainframe.

Table 56. Runtime configuration: emulator type parameters (Sheet 2 of 3)

Property	Description
Host port	Contains the port number for the Telnet 3270 protocol. The port number is usually **23**.
Short name	Can usually left as default. Some systems may require multiple sessions to be named **A**, **B**, etc.
Use TN3270E if available	Enables to use the extended protocol of the 3270E emulator for printing emulation purposes.
TN3270E device name	When the **Use TN3270E if available** parameter is set, specifies the device name for the printing session.
Support extended attributes	Used for compatibility between the converted screens and the current runtime. Clear check box if screens without extended attributes were converted.
Support extended data stream	Used for compatibility between the converted screens and the current runtime. Clear check box if screens without extended data stream were converted.
Host code page	The number and name of the EBCDIC code page on the host. This is used for choosing the EBCDIC/ANSI conversion table and during negotiation with the host.
Internal TN5250	
Host address	Contains the name or the IP address of the host iSeries.
Host port	Contains the port number for the Telnet 5250 protocol. The port number is usually **23**.
Short name	Can usually be left as default. Some systems may require multiple sessions to be named **A**, **B**, etc.
LU name	Specify the device (LU) name for the session. Verify that the OS/400 supports external LU name recognition.

Table 56. Runtime configuration: emulator type parameters (Sheet 3 of 3)

Property	Description
ASCII/EBCDIC conversion table	Choose the host application language.

Emulator Settings

Table 57. Runtime configuration: emulator settings parameters (Sheet 1 of 2)

Property	Description
25 message line appears in line 25	Directs the emulator to receive messages in line 25 of the host screen.
Session short name	Forces webMethods JIS to work with a given HLLAPI short name.
Field exit for any field	Automatic Field Exit for any field (not only numeric).
Host session color table name	Enters a color table to use for host session to override the green on black colors.
Minimum Host Quiet Time	Minimum amount of time (in milliseconds) the host screen must remain unchanged for the screen to be recognized. 0 - do no wait.
Maximum Host Quiet Time	Maximum amount of time (in milliseconds) to wait for the host screen to stabilize.
Display attributes	Option to display attributes in the host session.
Display DBCSSO/SI	When set, the host screen displays Shift-Out/shift-In characters. When cleared, these characters are replaced by blanks.
Erase EOF for all fields	Erases the remainder of a field using the ‘erase to End of field’ key instead of using blanks.

Table 57. Runtime configuration: emulator settings parameters (Sheet 2 of 2)

Property	Description
Dual Emulators	Determines whether webMethods JIS can work in dual emulator mode: GDS emulator and Telnet emulator. Note that this parameter is enabled only if you choose the GDS or the TN5250 emulators for **Emulator Type**.

Bleedthrough

Table 58. Runtime configuration: bleedthrough parameters

Property	Description
Use Just-in-time GUI	Option to use the Just-in-time GUI feature, if it is included in the Application. This option has no effect if the Application was compiled without Just-in-time GUI.

Display

Table 59. Runtime configuration: display parameters

Property	Description
Enable help on message line	Activates the **Help** button on the DIL.

XHTML

Table 60. Runtime configuration: display parameters (Sheet 1 of 3)

Property	Description
Runtime Directory	Specifies the location of the subapplication XHTML files that were created at runtime generation. Default location is `<JIS Root Directory>\classes\appls` Changes to this setting are effective only after server restart.
Images Location	Specifies the location of application image GIF files as they are accessed through the web server. For handling images through the internal Jetty HTTP Server, specify the relative path to the image location. Enter: `/classes/appls` Default: `http://localhost/classes/appls` Changes to this setting are effective only after server restart.

Javascript Location	Specifies the directory where the JIS Javascripts reside. This parameter works slightly differently depending on whether the application is running on the webMethods JIS proprietary server or on a J2EE application server.
	
	For the JIS Server:
	The default is
	`<Install directory>\classes\js`
	If you specify a value, \classes\js is automatically concatenated to the value.
	
	For a J2EE application server:
	`The default is`
	`<Install directory>\<applname>\js`
	If you specify a value, `\<applname>\js` is automatically concatenated to the value.
	
	Example: `JavascriptLocation=`
	`c:\appserver\domains`
	
	Changes to this setting are effective only after server restart.

Table 60. Runtime configuration: display parameters (Sheet 2 of 3)

Property	Description

Do HTML Merge	Specifies whether or not to enable HTML extensions.
	Unselected = disable HTML extensions
	Selected = enable HTML extensions
	Changes to this setting are effective only after server restart.

Merge Original	Specified whether to activate the merge of the original HTML files with the runtime HTML files, when no user templates are found. Changes to this setting are effective only after server restart.
DIL Position	Specifies whether the DIL messages appear at the top or the bottom of the Web page. The default is Bottom. Changes to this setting are effective only after server restart.
Default Button	Used to enable or disable default button functionality. Changes to this setting are effective only after server restart.
Show Table Up/Down Buttons	Specifies whether the Up and Down buttons are shown on tables. Changes to this setting are only effective after server restart.
Display Host Image When Out Of Sync	Specifies whether or not to show the host screen image if an Out-OfSync condition occurs. Changes to this setting are effective immediately.
Host Paging	Specifies whether the Page Up / Page Down keyboard events activate webMethods JIS's paging behavior or the browser's. Changes to this setting are effective only after server restart.
Use CSS Definitions	Specifies whether or not CSS definitions are to be used. Changes to this setting are effective only after server restart.
Fkey Support	Specifies whether or not Fkey support is enabled. Changes to this setting are effective only after server restart.
Keep Alive Interval in Seconds	Interval in seconds at which the JIS Server checks the connection with the Client. Changes to this setting are effective only after server restart.

Table 60. Runtime configuration: display parameters (Sheet 3 of 3)

Property	Description
Out Of Sync Handling	Specifies how the application is to behave when it encounters an “out-of-sync” condition. An out-of-sync condition is an error condition in which the screen being seen by the end-user does not represent the current host screen. • Page – Sends an “out-of-sync” message page to the user. • StatusLine – Sends the next page to the user, and add a message to the status line of the browser. • Sleep – Sends the next page to the user without informing him that an out-of-sync condition occurred. The default value for applications created with versions of webMethods JIS prior to version 9.0 is Page. The default for new applications under webMethods JIS version 9.0 and higher is StatusLine. Changes to this setting are effective immediately.
Popup Support	If enabled, when a host screen is identified as a popup window, a new browser window is open on top of the already existing window. The existing page is not refreshed, and all elements of the existing page are disabled.When a popup window is closed, the window beneath it is enabled. When the main window is closed, all open popup window are closed as well. Changes to this setting are effective only after server restart.
RMB Support	This setting indicates whether the Right Mouse Button popup menu feature is to be supported. • When unchecked: RMB popup menus not supported • When checked: RMB popup menus supported. For more information about Right Mouse Button popup support, see in Chapter 14 the section *Right Mouse Button Pop-Up Menus.* Changes to this setting are effective only after server restart.

Running the JIS Server as a Windows Service

You have the option of running the JIS Server as a Windows service. Two utilities are provided to help you accomplish this. One of the utilities registers the JIS Server in Windows' services database; the other utility is used by Windows to invoke the `jacadasv.bat` file used to start the JVM and load the server classes. There are a few new INI settings related to this procedure as well.

Registering the JIS Server in Windows

Use the utility `JBSToService.exe` to register the JIS Server in the Windows services database. The `JBSToService.exe` utility uses standard Win32 API calls, as well as direct registry access, for compatibility between different Windows versions. (e.g. Win2000 Server and Win2003 Server.)

`JBSToService.exe` accepts a variety of command line parameters by which you define the service's behavior. `JBSToService.exe` resides in the JIS root directory.

Parameters of JBSToService.exe

To list of all the parameters, simply run `JBSToService.exe` with no parameters. The following description is the same as that listed by the utility itself:

```
-c Create a new service

Create options:
        -i<Name of the service> (mandatory)
        -n<Displayed name of the service> (mandatory)
        -x<Full path to the executable file> (mandatory)
        -a Start the service automatically during system startup
        -m Start the service manually
        -l<Load ordering group of this service>
        -d<List of dependencies, separated by semicolons>
        -s<Account name for the service process>
        -p<Password of the account name>
        -h<Description of the service>

-r Remove a service

Remove options:
        -i<Name of the service> (mandatory)
```

The services database knows each service by a unique *service name*. This name is different than the *display name*, which is the name that appears in the Windows Services Application. To invoke to the Windows Services Application, from the Windows Start menu select **Programs > Administrative Tools > Services**.

To order to find the name of a service, right-click on its name, and choose "Properties". At the top of the properties dialog, you can see the service name. Just below it is the display name. The service name must be used whenever `JBSToService` is called to change the service's configuration. When removing a service, for instance, you need only supply this name.

The `-c` ("create") option is used for both creating and updating parameters. Repeatedly calling this utility, with different parameters, for the same service name, would apply the changes. However, note that parameters left out will be unchanged. Consider the following three lines

```
JBSToService.exe -c -iJac -a -h"First description" -n"JAC" <... other parameters...>
JBSToService.exe -c -iJac -a -h"Second description" -n"JIS Service" <... other parameters...>
JBSToService.exe -c -iJac <... other parameters...>
```

The result would be a service named `Jac`, with a display name JIS Service, and a description of Second description, and the service would be started manually. When the type of startup (`-a` or `-m`) is not explicitly specified, `-m` (manual startup) becomes the startup type by default.

The `-x` parameter specifies the executable that is launched when the service is started. When launching our JBS as a service, you must specify the full path to `JBSService.exe`, including the name of the executable itself.

A list of dependencies may be given, using a semicolon delimited list of service names. This list contains unique service names, *not* display names.

The rest of the parameters affect other settings of the service. Note that some of the parameters are mutually exclusive (`-a` and `-m`, for example) and the account name must be a valid account ("LocalSystem" is one default valid name).

More Examples of the Use of JBSToService.exe:

Create a service with a dependency list, starting automatically: `JBSToService.exe -c -iJac -a -n"JIS Service" -h"This is a test service" -d"Apache" -x"I:\guisysd\jbsservice.exe"`

Note: As of JIS 9.0.4 the JBSToService.exe command line has been considerably simplified. Refer to the JIS 9.0.4 release notes for more details.

Update the service, delete its dependency list:

```
JBSToService.exe -c -iJac -d""
```

Update the service, start manually:

```
JBSToService.exe -c -iJac -m
```

Update the Jac service, to use the INI file in a location I:\guisysd\jacadasv.ini, and change its description:

```
JBSToService.exe -c -iJac -a -n"JIS Service" -
x"\"I:\guisysd\jbsservice.exe\" -I:\guisysd\jacadasv.ini" -h"Description"
```

Remove the service:

```
JBSToService.exe -r -iJac
```

Caution

The `JBSToService.exe` utility uses Win32 API to process most of the parameters. `JBSToService.exe` does not perform any "reasonability check" on the values you choose. Using the `JBSToService.exe` utility carelessly could result in a damage to the operating system. Software AG is not responsible for any damage to your computer system as a result of improper usage of this utility.

Invoking the JIS Server as a Service

The `JBSService.exe` utility program reads the file `jacadasv.bat`, and creates a process from the command line written there. `JBSService.exe` uses I/O redirection to determine whether or not the JIS Server is running. If the JIS Server is running, `JBSService.exe` sends the string "quit" to the server's standard input, in order to stop it. This guarantees that all of the JIS Server's processes are down.

The `JBSService.exe` utility should ***not*** be called directly. Rather, it should be registered as the executable file for the JIS Server, by the `JBSToService` utility described above.

There are a few configuration parameters that must be set before running `JBSService.exe.` The parameters must be saved in a special INI file named `JBSService.ini`. The full list of parameters is described in the INI parameters section. Below is an example of a minimal INI file.

```
[JBSService]
ServiceName=JISService
CommandFile=C:\Ace\jacadasv.bat
LaunchFolder=C:\Ace
```

You may, however, place the INI file in any folder, provided you use the special syntax for the executable file (the **`-x`** parameter in `JBSToService.exe`). The syntax, is to pass the INI full name and path to the `JBSService.exe`, as a command line parameter. For instance, the following command line:

```
C:\Ace\JBSService.exe -C:\Ace\JacadaFiles\classes\
jacadasv.ini
```

instructs JBSService to extract the parameters from the `jacadasv.ini` file. In this case, you should edit `jacadasv.ini` file and add the `[JBSService]` section. In order to use such a command line, you should use the double-quotes syntax for the `-x` parameter.

The following examples demonstrate this:

No quotes needed, name without spaces:

```
JBSService -iJac -xC:\Ace\JBSService.exe
```

Quotes needed, because of the space character in "Program Files": `JBSService -iJac -x"C:\`**`Program Files`**`\JBSService.exe"`

Two sets of quotes needed:

```
JBSService -iJac -x"\"C:\Program Files\JBSService.exe\" -
C:\Ace\JacadaFiles\jacadasv.ini"
```

In this example, one outer pair of quotes is used in order to pass the whole string `C:\...jacadasv.ini` as a single parameter. The inner (escaped) pair is used, in order to register a command line with one executable file and one parameter. When inspecting the properties of this service, the "Path To Executable" box would contain exactly

```
"C:\Program Files\JBSService.exe" -C:\Ace\JacadaFiles\jacadasv.ini
```

A final example:

```
JBSService -iJac -x"\"C:\Program Files\JBSService.exe\" -C:\Program
Files\JacadaFiles\jacadasv.ini"
```

This example is the same as the previous one, except that in this case the `jacadasv.ini` is placed under a long name folder hierarchy (`Program Files`). Note that unlike the executable name, you do not need to double quote this name.

Log File

The log file of `JBSService.exe` contains important information, for debugging a launching failure. It dumps the INI file name, the launching file and the home folder, as well as the Java command (if `jacadasv.bat` is found), and the server's input (e.g. "STARTED").

Note that the file `jacadasv.bat` contains wildcards for additional parameters (for example, %1 %2). These should be removed. You may place some default command line parameters, but you need to erase all of the %n's. It is best to save a copy of the original batch file so that you can easily revert to the original if necessary.

Logging off from the machine

It is common to start a list of services, using the administrator's login, and then logoff and let the services run. However, the JVM normally terminates all of its processes at logoff. In order to avoid this behavior, add the JVM parameter `-Xrs` to both the `jacadasv.bat` file and `jacadasv.ini` file.

```
[VMCommandLine]
JavaOptions=-Xrs -Djava.security.policy=$RootDir\classes\jacadasv.policy
```

Printing Emulation

This section introduces the Printer Emulator and explains how to set it up and use it. The Printer Emulator enables your end-users to send printing requests to the host and receive the printing results on their own computer.

This section covers the following topics:

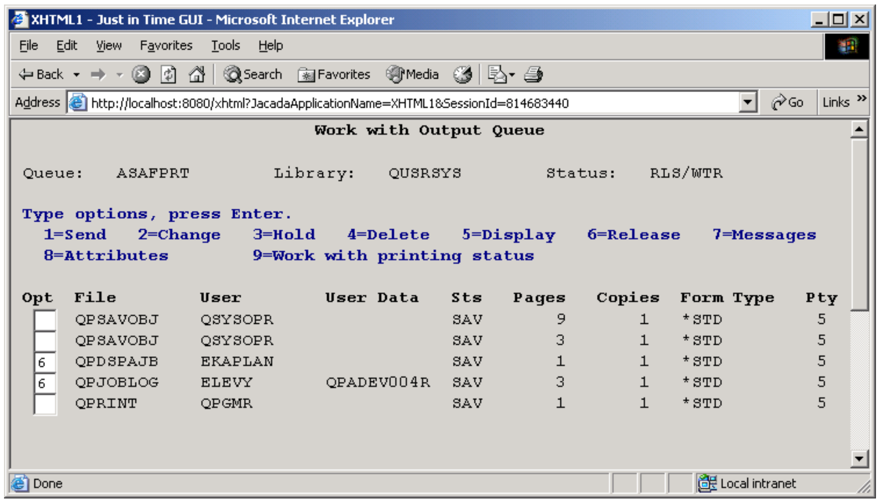
- Sending a Printing Request to the Host
- Viewing Print Job Results
- Enabling Printing Emulation on the JIS Server

Sending a Printing Request to the Host

During runtime, an end-user can send a printing request to the host. Sending a printing request to the host is done as you would normally do so on a green-screen dumb terminal.

To send a printing request to the host:

1. Browse through the host screens to the screen which enables host printing.
2. Next to the desired host file, or files, insert the proper selection for host printing, as seen in this example:


XHTML1 - Just in Time GUI - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost:8080/xhtml?JacadaApplicationName=XHTML1&SessionId=814683440
Go Links
Done
Local intranet

```
                          Work with Output Queue

Queue:    ASAFPRT        Library:    QUSRSYS        Status:    RLS/WTR

Type options, press Enter.
  1=Send    2=Change    3=Hold    4=Delete    5=Display    6=Release    7=Messages
  8=Attributes          9=Work with printing status

Opt  File        User        User Data    Sts    Pages    Copies   Form Type   Pty
     QPSAVOBJ    QSYSOPR                  SAV        9         1   *STD          5
     QPSAVOBJ    QSYSOPR                  SAV        3         1   *STD          5
 6   QPDSPAJB    EKAPLAN                  SAV        1         1   *STD          5
 6   QPJOBLOG    ELEVY       QPADEV004R   SAV        3         1   *STD          5
     QPRINT      QPGMR                    SAV        1         1   *STD          5
```

Figure 44. Sending a printing request

3 Press **OK** to submit the request to the host.

The outcome of this request is sent to the JIS Server's printing emulator. Upon request, the JIS Server sends the outcome of this request to the end-user.

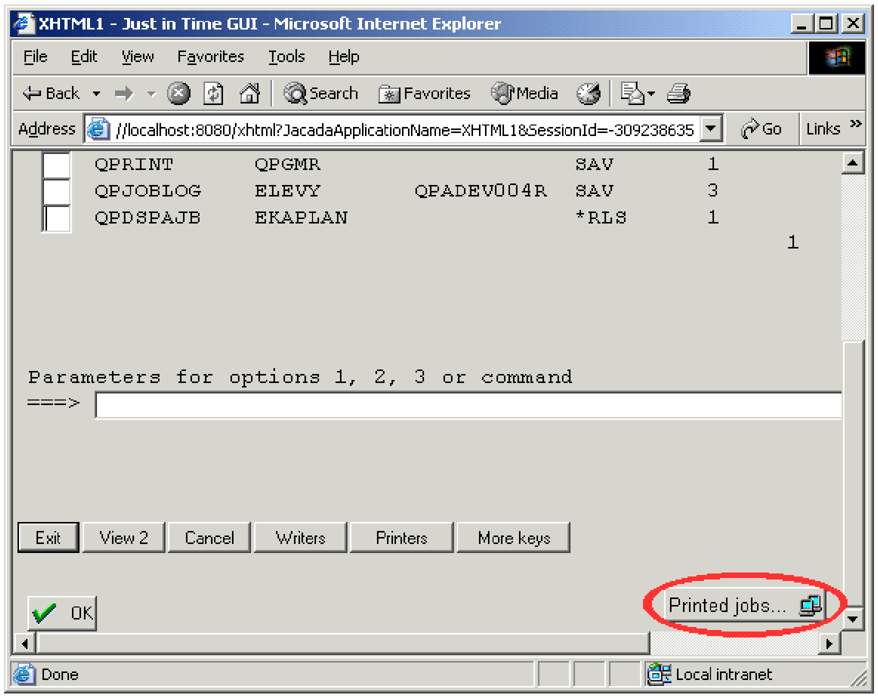
Viewing Print Job Results

Once printing jobs have been sent to the host, the end-user can view them. Upon request, a new browser window opens on the end-user's computer displaying the print jobs performed by the host. From these, the end-user can select and print any desired print job.

To view a print job result:

1 Perform a Host Printing request as seen on the preceding page.

2 Click the **Printed Jobs** button, located on the bottom right of each screen:


XHTML1 - Just in Time GUI - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address //localhost:8080/xhtml?JacadaApplicationName=XHTML1&SessionId=-309238635
Go Links

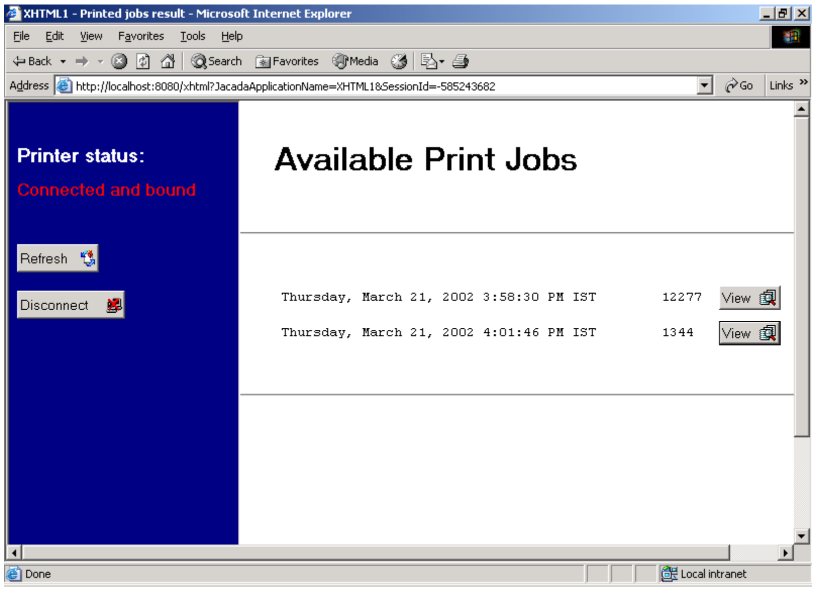
```
QPRINT     QPGMR                SAV     1
QPJOBLOG   ELEVY    QPADEV004R  SAV     3
QPDSPAJB   EKAPLAN              *RLS    1
                                              1

Parameters for options 1, 2, 3 or command
===>
```

Exit
View 2
Cancel
Writers
Printers
More keys
OK
Printed jobs...
Done
Local intranet

Figure 45. Printed jobs button

Doing this causes a new window to open, without disrupting the flow of the Application. This window is the **Available Print Jobs** window.

The **Available Print Jobs** window displays a list of the printing job results:


XHTML1 - Printed jobs result - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost:8080/xhtml?JacadaApplicationName=XHTML1&SessionId=-585243682
Go Links
Refresh
Disconnect
View
View
Done
Local intranet

Printer status:
Connected and bound

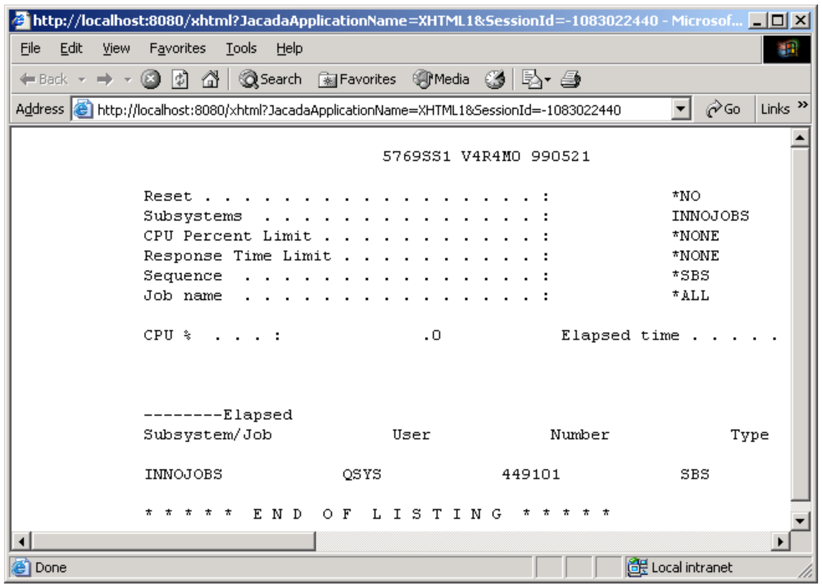
Available Print Jobs

```
Thursday, March 21, 2002 3:58:30 PM IST     12277
Thursday, March 21, 2002 4:01:46 PM IST     1344
```

Figure 46. Available print jobs window

3 Click the **View** button next to the desired print job.

The browser displays an HTML page with the contents of the printing request:


http://localhost:8080/xhtml?JacadaApplicationName=XHTML1&SessionId=-1083022440 - Microsof...
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost:8080/xhtml?JacadaApplicationName=XHTML1&SessionId=-1083022440
Go Links
Done
Local intranet

```
                         5769SS1 V4R4M0 990521

             Reset . . . . . . . . . . . . . . . . . :            *NO
             Subsystems  . . . . . . . . . . . . . . :            INNOJOBS
             CPU Percent Limit . . . . . . . . . . . :            *NONE
             Response Time Limit . . . . . . . . . . :            *NONE
             Sequence  . . . . . . . . . . . . . . . :            *SBS
             Job name  . . . . . . . . . . . . . . . :            *ALL

             CPU %  . . . :            .0           Elapsed time . . . . .



             --------Elapsed
             Subsystem/Job          User           Number          Type

             INNOJOBS          QSYS           449101          SBS

             * * * * *  E N D  O F  L I S T I N G  * * * * *
```

Figure 47. The outcome of the printing request

4 To print the outcome, select **File > Print**.

The Available Print Jobs Window

In this window, you can do the following:

- View a print job, by clicking the **View** button next to the desired print job.
- Update the contents of the screen, by clicking the **Refresh** button. As you send more printing jobs to the host, you need to update the **Available Print Jobs** window.
- Connect or Disconnect from the host LUName, by clicking either the **Connect** button or the **Disconnect** button. This disables printing job viewing.

Chapter 6. Language Localization

The language localization feature enables an application running on the Server in one language to be displayed by the client in any other language, or simultaneously by several clients using different languages. Moreover, this feature can be customized to display regional variations and specific professional jargon.

The following topics are described:

How the Localization Feature Works

The localization feature has two main advantages: first, a legacy application designated to be used by a multi-lingual clientele will only have to be converted once, to a GUI representing the original language of the host application. An end-user who wishes to use the GUI application in a different language needs only to be provided with the translation of the strings present in the original application. Secondly, only one runtime, in the original language, needs to be installed on the server. This runtime then serves as the basis for running applications in other languages.

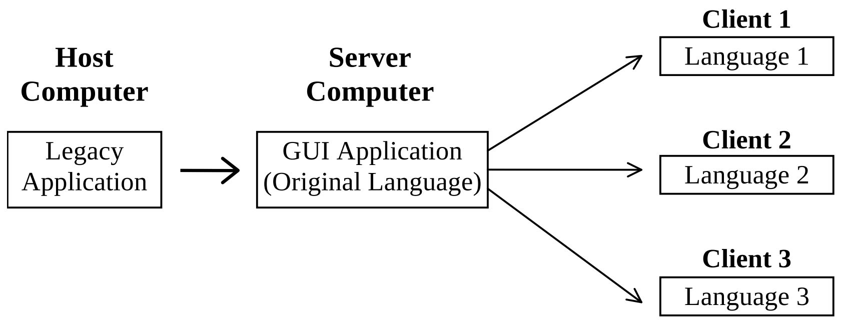


Figure 48. Language localization of JIS applications

The Localization Feature Workflow

When using the Localization feature, the client displays strings translated into the desired language, instead of displaying the application's original strings. These translated strings are imported into the application from external resources during runtime.

1 During the compilation process, a resource file is created and placed in `<JavaRootDir>\JacadaFiles\classes\appls\<ApplName>\resources\`

  This file lists all the original static strings gathered from all the Subapplications making up a library or an application.

2 For each desired language, make a copy of the resource file. These files become the translated resource files.

3 Translate the original strings into the desired language(s).

4 Add the translated strings to the appropriate translated resource file.

What Happens During Runtime:

1 The appropriate translated resource file is either preloaded with the library classes through a JAR file or when the client places its first request for the string information.

2 The Application runs displaying the translated strings.

Activating the Localization feature

The Localization feature is predefined as disabled.

To activate the Localization feature:

1 In the JIS converter, from the **Options** menu, select **Runtime Generation Options**.

2 Set the **Client Language Localization** check box.

3 Generate a full runtime.

During the compilation process a text file containing all the application's strings is generated. The strings contained in this file appear in the original host application language and form the base for translation into other languages.

The translated resource files contain a series of Key and Value pairs. The original language string acts as the key and the translated string is the value.

The Resource Files

The Localization feature relies upon two resource files:

- An **Original** resource file which is created during compilation
- A **Translated** Resource file which should be created for each additional language.

The Original Resource File

When you generate a runtime, a text file named `StringResource.res` is generated and is placed in the following directory:

```
<JavaRootDir>\JacadaFiles\classes\appls\<ApplName>\resources\
StringResource.res
```

This file is created based on the collective input of all the Subapplication-specific string resource files. The file contains a column of key strings in quotation marks followed by a space and an equal sign: `“<original string>”[<space>]=`

Example: `"Hello World" ="Goodbye World"`

The Translated Resource File

After the creation of the original resource file, proceed as follows:

1 Make one copy of the `StringResource.res` file for each desired language. The file is named according to the language using the system, and should be written in the following format:

```
StringResource_<locale code>.res
```

Note: Using the Java standard Locale code is recommended.

2 Append the translated strings to the original language strings in the following manner:
`“<key>”[<space>]=[<space>]”<value>”<line break>`

Example5. The resource file for Canadian French

The translated resource file for Canadian French is named

```
StringResource_fr_CA.res
```

Within this file, the original and translated language strings appear as follows:

```
“Hello World” = “Bonjour le Monde”
```

```
“Goodbye World” = “Au revoir le Monde”
```

There can be only one pair on each line. Blank lines are ignored as are lines starting with a double slash (//). Take this into account when writing comments or commenting out strings.

Note: The size of each control in the GUI does not increase to accommodate the length of the translated strings. If a string exceeds the allocated space it is truncated.
When creating a control, or when allotting space for a field during the conversion, make allowance for eventual longer translated strings. When translating resources, be aware of the length limitation.
In runtime, the original language strings are used for missing or untranslated entries.
If you need to add special characters such as a quotation mark (”) in your translation, use the Java escape sequence convention.

Resource Maintenance

The translation of the string resource is performed after completing application development. The generated string resource serves as the basis for the creation of resources in the desired languages. When modifications are made to the application at a later stage, a new list is generated. The translated resources thus lose their fidelity to the application. It is necessary then to merge the list that contains the translated string with the new and untranslated strings that were generated in the new resource. This is currently done manually.

Setting the Runtime Localization Mechanism

The localization parameter is transferred to the server via a URL parameter.

Let’s say that you have translated the static strings in your Application into Canadian French and named the translated resource file `StringResource_fr_CA.res`. By adding the following string to the Application HTML file, your runtime is displayed translated into Canadian French:

```
src="/Xhtml?JacadaApplicationName=<ApplName>&Language=fr_ca"
```

If no resource file named `StringResource_fr_CA.res` exists, the runtime displays the original strings.

String Types Handled by Localization

The localization feature supports a large array of static string types. Below find a list of place holders from which static strings are gathered.

- Window captions
- Main Window (application independent strings)
- Subapplication
- Bubble help
- Labels
- Tab controls
- Button / check box / option box labels
- HTML—from client, server
- Message Box—from host, client, server
- **About** dialog box text and buttons

Note: You may inadvertently change the status of a string while modifying a Subapplication in the converter. Changing a string's status from static to dynamic or vice versa will also result in changes in the newly generated resource file. Static strings are written to the resource file, dynamic ones are not.

ISO Language and Country Codes

Although not mandatory, we recommend that you use the two letter language code and country code standard when naming your translated resource files. The codes are derived from the ISO 639 standard (for language code) and the ISO 3166 standard (for country code). Complete lists can be easily found over the Internet. Table 61 constitutes an example of some locales.

Table 61. International standards for localization (Sheet 1 of 2)

Language Name	Language Code	Country code
English (Australian)	en	AU
English (Canadian)	en	CA
English (United States)	en	US

Table 61. International standards for localization (Sheet 2 of 2)

Language Name	Language Code	Country code
French (Swiss)	fr	CH
French (France)	fr	FR
Spanish (Spain)	es	ES

Current Limitations

This feature supports localization of static strings only, it does not attempt to translate variable field values. Use the server's dictionary to translate variable fields.

- DIL messages can pass localization. However, the string that appears in the DIL is not automatically added to the original resource file `StringResource.res`. You must add the original and translated strings to their appropriate files manually.
- For the application-independent JIS string resources, only partial translations are provided.
- Unicode escape sequences in resources are not supported.
- Language localization is not implemented on IBM's NC. This is due to a bug in its NSM version 2.
- In XHTML, dates are formatted according to the locale of the end-user's machine. Therefore, dates are not affected by the Localization feature.
- Decimal points are not supported by the XHTML client and therefore are not affected by Localization.
- HTML error pages cannot be localized. You can, however, write an error page using JavaScript that finds the end-user's locale and translates the HTML page according to this locale.

Chapter 7. XHTML Runtime Architecture

The core of the XHTML runtime is in the JIS Server. As already discussed earlier in this manual, the JIS Server is a component that mediates between the clients and the host application. In order to understand the XHTML runtime architecture, it is necessary to examine the structure of the JIS Server and the XHTML Processing Module within the server. XHTML processing is reviewed in detail, including references to how user HTML and Java extensions can be incorporated into the runtime application.

The following topics are discussed:

- "JIS Server Components" on page 233
- "XHTML Processing Module" on page 234
- "XHTML Processing" on page 236

JIS Server Components

The diagram below illustrates the general flow of information between the host, the JIS Server and the XHTML client.

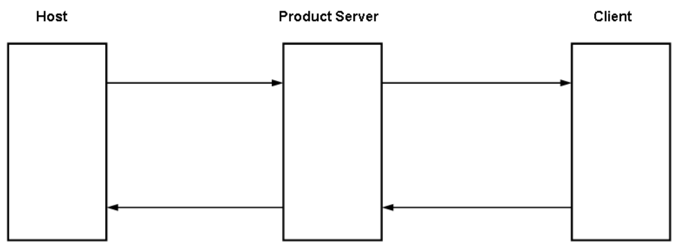


Figure 49. Information flow in runtime

In order to understand the XHTML runtime architecture, it is necessary to examine the structure of the JIS Server. The JIS Server is composed of the following components:

- Server Logic
- XHTML Processing Module
- HTTP Support

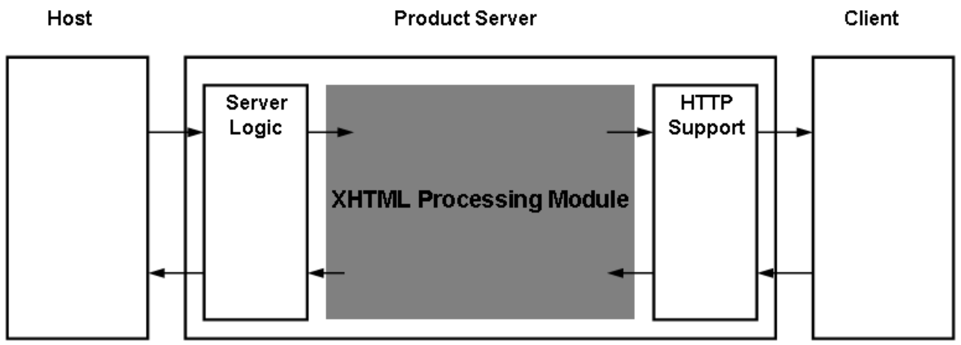


Figure 50. JIS Server components

Note: The relative size of the components in all diagrams is not necessarily proportionate to their real size. The XHTML Processing Module and its components have been enlarged for the sake of clarity.

The **Server Logic** component is responsible for communicating with the host. This includes performing tasks such as identifying the current screen received from the host, building the current Subapplication, managing variables, sending Subapplication data to the host, and more.

The **HTTP Support** is built into the JIS Server to enable clients to communicate with the JIS Server using HTTP.

The **XHTML Processing Module** resides between the Server Logic and the HTTP Support. This module is responsible both for sending the current Subapplication XHTML to the client, and then receiving the user data and conveying it back to the host. These processes include incorporating user HTML extensions and activating Java extensions.

This chapter describes in detail the various processes that take place in the XHTML Processing Module. The details are exposed in a gradual manner. Each section focuses on a different part of the XHTML Processing module. The accompanying diagrams highlight the components under discussion to help you concentrate on them. The other components remain greyed out, serving as a reference.

XHTML Processing Module

The XHTML Processing Module in the JIS Server is responsible both for receiving the user data and conveying it back to the server logic, and for building the current Subapplication XHTML to send to the client.

The XHTML Processing Module is composed of two parts: one that is responsible for the on-going processing between the client and the server logic, and another component that generates and caches the static XHTMLs. The static XHTMLs are generated once per Subapplication and per browser type, and then each XHTML is retrieved from the cache by the on-going processing as needed.

The structure of the XHTML Processing Module is shown in the following diagram:

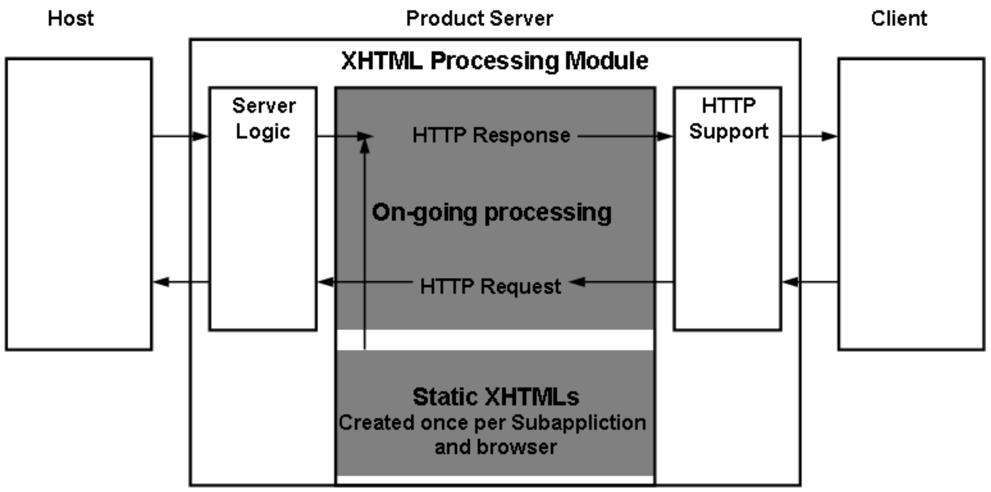


Figure 51. The XHTML processing module

HTTP Request Processing - Client to Host

The on-going processing between the client and the host involves receiving the user data from the client, implementing Java extensions, if in use, and conveying this information, via the server logic, to the host.

HTTP Response Processing - Host to Client

The on-going processing between the server logic and the client involves building the current Subapplication XHTML using:

- Static XHTMLs stored on the JIS Server, which include HTML user extensions, if in use.
- Dynamic Subapplication information, received from the host.
- Implementing Java extensions, if in use.

XHTML Processing

The following sections provide more details on XHTML processing that takes place within the XHTML Processing Module, including the following topics:

- Static and dynamic Subapplication information.
- On-going processing between the host and the client.
- Building the static XHTML, including HTML extensions.
- Updating the static XHTML with dynamic information.
- Writing Java extensions.

Static and Dynamic Subapplication Information

The current Subapplication XHTML contains two types of data: static information and dynamic information.

The static information includes both content that is constant, such as headers, as well as general characteristics of the HTML page, such as its fonts. The static information is generated by the Runtime Generation process in ACE, and then updated during runtime in the JIS Server with runtime settings and HTML extensions if they are being used.

Dynamic information, on the other hand, depends on the current contents of the current host screen. For example, the contents of an employee details screen change according to each specific employee.

The HTML page viewed by the client, must contain the combination of the static and the dynamic Subapplication data.

On-going Processing Between Host and Client

The following diagram focuses on the on-going processing between the client and the host, and the host back to the client, after an initial connection has already been established:

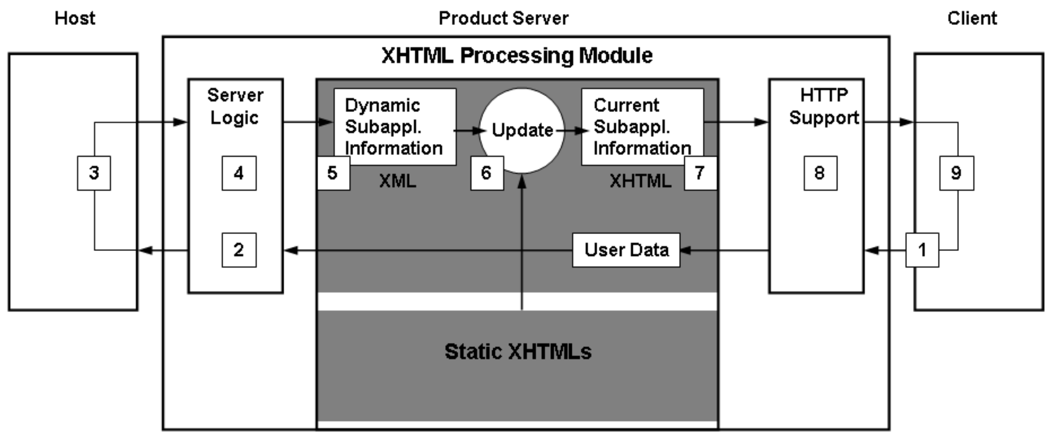


Figure 52. Client-host information flow

The steps below describe a cycle of a user sending an HTTP request to the host, and then receiving an XHTML from the host as the HTTP response:

1 The user updates the current form and submits it as an HTTP request.
2 The user data is assigned to fields and sent back to the host, via the Server Logic component.
3 The host application reacts to the user data received from the client, and as a result displays the suitable screen.
4 The Server Logic component identifies the current Subapplication and sends the dynamic runtime information to the XHTML Processing Module.
5 The XHTML Processing Module creates an XML with the dynamic Subapplication information.
6 The Update process retrieves the current Subapplication's static XHTML and updates it with the dynamic information from the XML.
7 The result of the Update process is an updated XHTML for the current Subapplication.
8 The updated XHTML is sent, as the HTTP response, to the client.
9 The page is displayed in the client browser. The cycle begins once again (see step 1).

Building a Static XHTML with HTML Extensions

The XHTML Processing Module contains a component that generates and caches an Application's static XHTMLs. Within the lifetime of a JIS Server, a static XHTML is created once for each Subapplication, according to the browser type currently in use. Each time the JIS Server encounters a new combination of a Subapplication and browser type, the static XHTML is generated and added to the cache, for future use. The static XHTML is created using a base XHTML, and then merging any user HTML extensions.

Building the Base XHTML

The base XHTML is created using the following input:

- The Subapplication XML. This XML was created during the Runtime Generation process in ACE.
- The browser XSL. Each browser type has its own XSL.
- The runtime settings that appear in the jacadasv.ini file. For example, the location of the DIL (Dynamic Information Line).

The XSL that corresponds to the current client browser is applied to the current Subapplication XML, taking into account the runtime settings in the jacadasv.ini.

The following diagram illustrates the creation of the base XHTML:

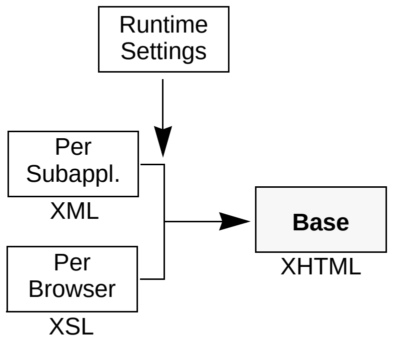


Figure 53. Base XHTML generation

Merging HTML Extensions with the Base XHTML

When user HTML extensions are in use, they need to be merged with the base XHTML, in order to create the static XHTML. After merging any extensions, the static XHTML is cached on the JIS Server.

Updating the Static XHTML with Dynamic Information

The following diagram illustrates how the static XHTML is updated with the dynamic information received from the host:

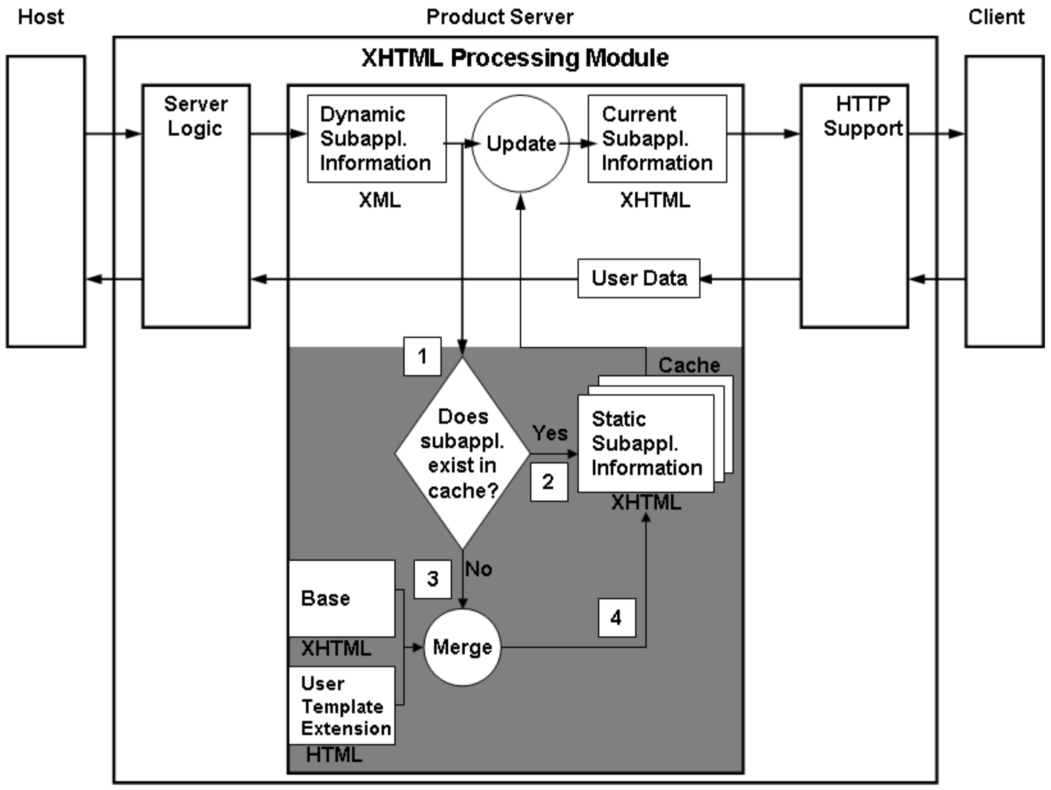


Figure 54. Update of dynamic XHTML

The following steps take place in the XHTML processing module:

1 After an XML is created from the dynamic Subapplication information, the static XHTML is searched for in the cache, according to the Subapplication and the client browser type.

2 If this combination already exists in the cache, the corresponding XHTML is retrieved.

3 Otherwise, the XHTML needs to be created by merging between the base XHTML and the user template extension HTML, if it exists. Once merged, the XHTML is retrieved for the current Subapplication and then added to the cache.

Writing Java Extensions

In addition to HTML extensions, it is also possible to write Java extensions, using events. These events are activated during the on-going processing between the server logic and the client. The first two events are activated during the process of building the XHTML to send to the client, whereas the last event is activated after the client submits the form:

- `onControlReady`

- `onPageLoad`
- `onPageSubmit`

The **`onControlReady`** event enables you to write extensions that affect a particular control type across the entire Application. For example, replace all the date controls in the Application with a customized date control. This event occurs during the updating of the static XHTML with the dynamic runtime information. For each control, after it is ready, if a Java extension exists for this control type, it is applied. Control-level extensions are Application independent, and can be used to apply “skins” across different Applications.

The **`onPageLoad`** event is activated after the page has been updated and before it is sent to the client. This event enables you to control the contents and the look of the HTML page. For example, you can set control properties such as size, location, text, font and color, etc. You can also add or delete controls, add JavaScript functions, and more.

The **`onPageSubmit`** event occurs immediately after the client submits the form and before any server processing operations, such as formatting or applying dictionaries, are done on the user data. The `onPageSubmit` event can be used to manipulate the user data. For example, to write Java code that converts user data to the required host format, by removing masking; or to save the user data to an external database. Note that you cannot use the `onPageSubmit` event to stop the submission of the page.

Both the `onPageLoad` and `onPageSubmit` events are specific to an Application, and can be used to write extensions for a Subapplication, a library or an Application. These extensions are activated in a hierarchical manner, and Java inheritance exists between these classes.

The following diagram illustrates when these events take place within the XHTML Processing Module:

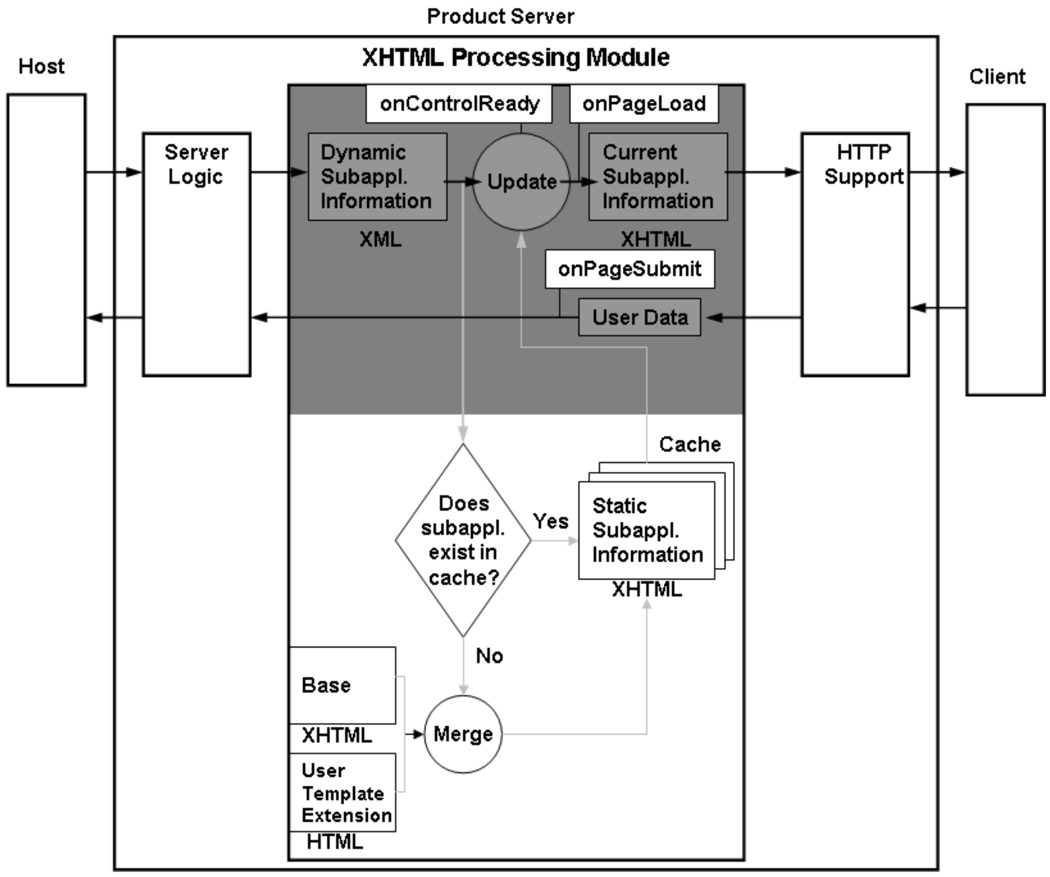


Figure 55. Event activation

Chapter 8. Enhancing Your Application Using HTML Extensions

After developing your Application in ACE, it is still possible to improve your Application's look and feel. Such improvements are made outside of ACE and are manually incorporated into the JIS Application. This can be done by creating HTML extensions. During runtime, the JIS Server merges these HTML extensions with your runtime Application. User-created HTML extensions enable you to incorporate JavaScripts, VBScripts and various other HTML features into runtime-generated XHTMLs.

This chapter discusses HTML extensions, teaches how to create them, and provides several examples of the added functionality that they can provide.

The following topics are discussed:

- "HTML Extensions" on page 243
- "Creating HTML Extensions" on page 248
- "The OutOfSync Screen" on page 250
- "Using JavaScript in HTML Extensions" on page 252
- "Examples of User HTML Extensions" on page 254
- "Customizing HTML Error Pages" on page 258
- "CSS Usage in the JIS XHTML Client" on page 267

HTML Extensions

In the JIS Server, the XHTML Processing Module contains a component that generates and caches an Application's static XHTMLs. Within the lifetime of a JIS Server, a static XHTML is created once for each Subapplication, according to the browser type currently in use, and then it is cached on the JIS Server.

The static XHTML is created by merging a base XHTML with a user HTML template extension. Then, the static XHTML is updated with the dynamic information received from the host. The following diagram illustrates these processes in the XHTML Processing Module:

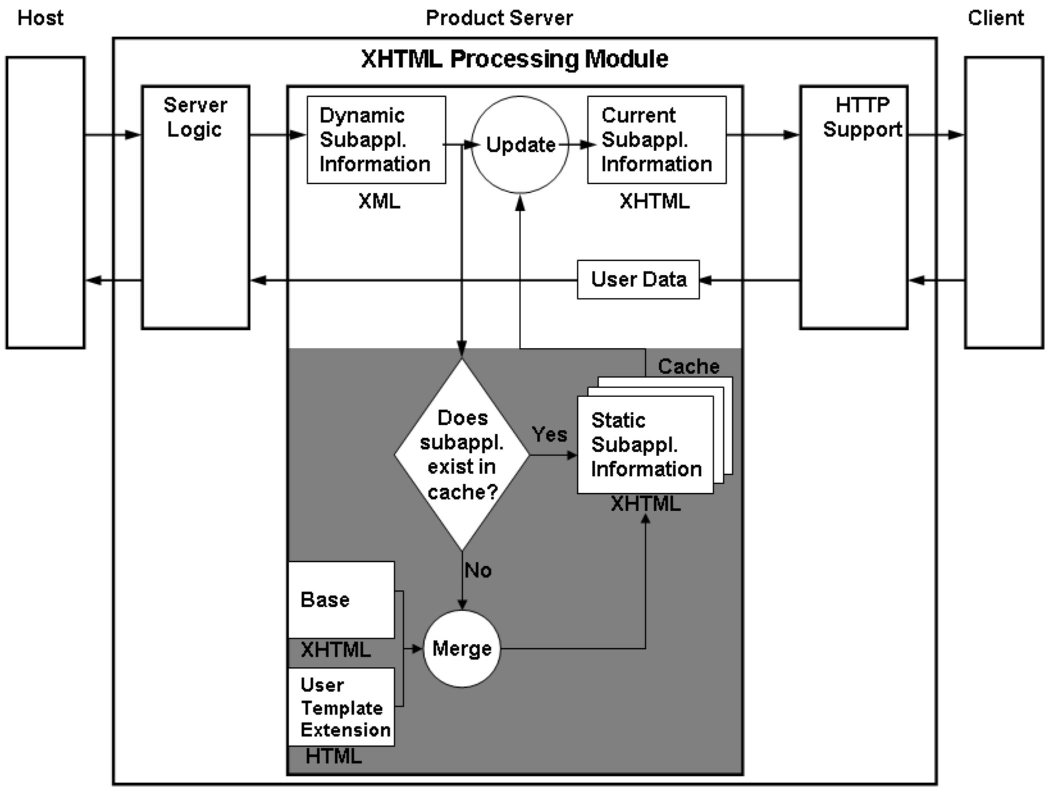


Figure 56. XHTML generation

Note: For more information, see Chapter 7 - "XHTML Runtime Architecture" on page 233.

Location of Files on the JIS Server

The *original* runtime-generated XHTMLs and the *user* HTML extension files are each located under separate directories under on the server machine, as shown in Figure 57:

- The *original* directory contains the original runtime-generated XHTMLs.
- The *user* directory contains the user HTML extension files.

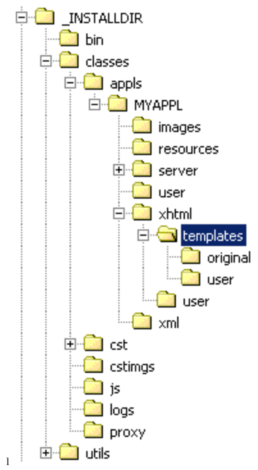


Figure 57. HTML-containing directories

During runtime, the JIS Server knows to look in the *user* directory for extension files and incorporate them into the XHTMLs that it sends to the Client. If no extension files are found in the *user* directory, the JIS Server takes the files from the *original* directory.

Converting User HTML Extension Files to DOM XHTMLs

During runtime, when the JIS Server locates a user HTML extension file, using the HTML Syntax Checker, it first "cleans up" the extension file converting it to standardized DOM XHTML format.

The HTML Syntax Checker performs the following tasks:

- Detecting and correcting missing, mismatched or misplaced end tags.
- Fixing problems with heading emphasis.
- Placing <HR> horizontal line tags in the right position.
- Adding missing quotes around attribute values.
- Recovering from mixed up tags.
- Perfecting lists by putting in tags missed out.

After a user HTML extension file is standardized and converted to DOM XHTML, the JIS Server can merge the converted extension file with its corresponding base XHTML.

Merging the XHTMLs Into One File

After the *user* HTML extension file is converted to XHTML, the JIS Server merges HTML extension file with its corresponding base XHTML.

The merging is carried out according to the following rules:

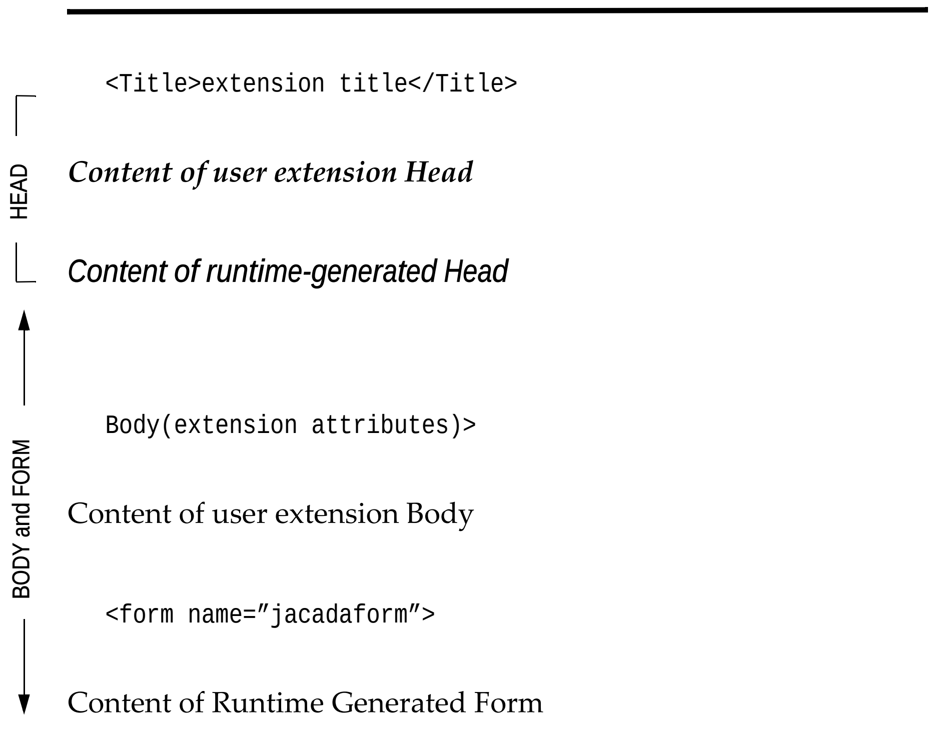
- The *user* file's `<Body>` tag attributes override the runtime-generated file's `<Body>` tag attributes.
- The *user* file's `<Title>` tag overrides the *original* file's `<Title>` tag.
- The runtime-generated file's `<jacadaform>` and `<jacadaprinting>` tag overrides the *user* file's `<jacadaform>` and `<jacadaprinting>` tag.

Note: In the runtime generated file, there are two form tags called <*jacadaform*> and <*jacadaprinting*>. When the user and runtime generated files are merged, these tags are added to the user file. Do not try to add a <*jacadaform*> *or a* <*jacadaprinting*> to the user file, since the tags in the runtime generated file take precedence and your changes will be overwritten when the user and runtime generated files are merged.

The Merged XHTML

The merged XHTML file is structured as follows:

`DOCTYPE line`



```
<Title>extension title</Title>
```

Content of user extension Head

Content of runtime-generated Head

```
Body(extension attributes)>
```

Content of user extension Body

```
<form name="jacadaform">
```

Content of Runtime Generated Form

```
</form>

</Body>
```

The Merging Process During Development

When developing the Application in ACE, you can incorporate HTML extensions into your Application, in order to test the runtime Application before distribution.

To incorporate an HTML extension file for testing while working in the development environment, you need to place it in the Application's XHTML user templates directory - `<InstallDir>\JacadaFiles\classes\appls\<ApplName>\xhtml\templates\`**user**

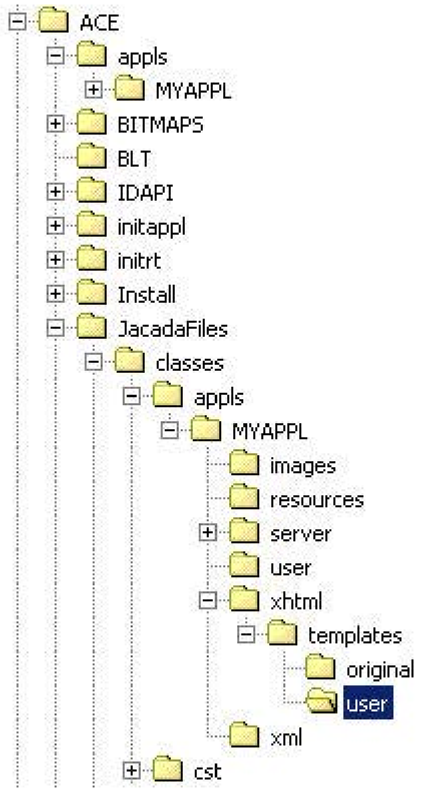


Figure 58. user extensions directory

Later, when you test your runtime on the development machine, the JIS Server merges the *user* or *original* XHTMLs with dynamic XHTMLs created during runtime.

Note: The JIS Server first searches for files in the user directory, and then searches the original directory.

Creating HTML Extensions

To create XHTML Extensions you do the following:

1. Configure the JIS Server to enable HTML extensions.
2. Write the user HTML extension file.
3. Incorporate the extension into your Application.

Configuring the JIS Server to Enable HTML Extensions

This is done in the JIS Server INI file.

To configure the JIS Server to enable HTML extensions:

1. Open the JIS Server INI file, `jacadasv.ini`, in a text editor. During development, the file is located in the following directory: `<InstallDir>\JacadaFiles\Classes\`

   During deployment, the file is located in the following directory: <InstallDir>\Classes\
2. Under the `[XHTML]` section, find the `DoHTMLMerge=` parameter. Give the `DoHTMLMerge=` parameter the value `1`. The `[XHTML]` section should read as follows:

   ```
   [XHTML]
   RuntimeDirectory=f:\ACE\JacadaFiles\classes\appls
   ImagesLocation=http://10.10.10.10/jacada/classes/appls
   DoHTMLMerge=1
   XSLDriver=XT
   DILPosition=bottom
   ```
3. Save and close the file.

Writing a User HTML Extensions File

The user HTML extension file can be any HTML or XHTML file. When you write the file, keep in mind the following points:

- Do not try to add `<jacadaform>` or `<jacadaprinting>` tags to the *user* file, since the tags in the runtime generated file take precedence and your changes will be overwritten when the user and runtime generated files are merged.

- The title of the *user* HTML wins. If no title is specified in the *user* HTML, the title of the JIS-generated HTML is taken instead.
- The attributes of the `<BODY>` tag of the *user* HTML win. If, for example, no background color is specified in the *user* HTML, the runtime-generated HTML color is taken.
- The merging process does not require the *user* HTML page or tags to be in a certain case. The XHTML tags of the outcome of the merging process are in lower case. Contents of the XHTML, tag values and strings, remain in their original case.
- The format of the output of the merging is always XHTML, even if the *user* HTML is not well-formed.

Naming and Placing a User HTML Extension File

It is important to correctly name and position your user HTML extension file. The file's name has a direct bearing on the level at which the extension file is incorporated into the Application.

Extension files can exist on one of three levels:

- Subapplication Level
- Library Level
- Application Level

These three levels appear in order of priority. Subapplication level extensions take the highest priority and override both Library and Application level extensions. Library level extensions take priority and override Application level extensions. Application level extensions have the lowest level of priority.

Application Level

To merge a user extension file with the whole Application, you must name it `"appl.html"` and save it in the JIS Server's *user* directory. The extension file is then incorporated into the whole Application and is set into all of the screens.

Library Level

To merge a user extension file with a whole library you must also give it the name `"appl.html"`. You must place the file in the library's user directory.

Subapplication Level

To merge a user extension file with a specific Subapplication, you must give it the name of the Subapplication. If the Subapplication's name is `"MBF005"`, then you must name your extension file `"MBF005.htm"` and save it in the JIS Server's *user* directory. The extension file is then incorporated only into that one screen.

Incorporating an Extension File into Your Application

The process of incorporating extension files into your Application is different, depending on whether you are deploying the runtime Application or still developing the Application in ACE.

During Deployment

To incorporate a user extension into your Application during deployment:

1. Place the HTML file in the following directory on the server machine:
   `<InstallDir>\classes\appls\<ApplName>\xhtml\templates\user`
2. Make sure that the file bears the correct name.
3. Restart the JIS Server.

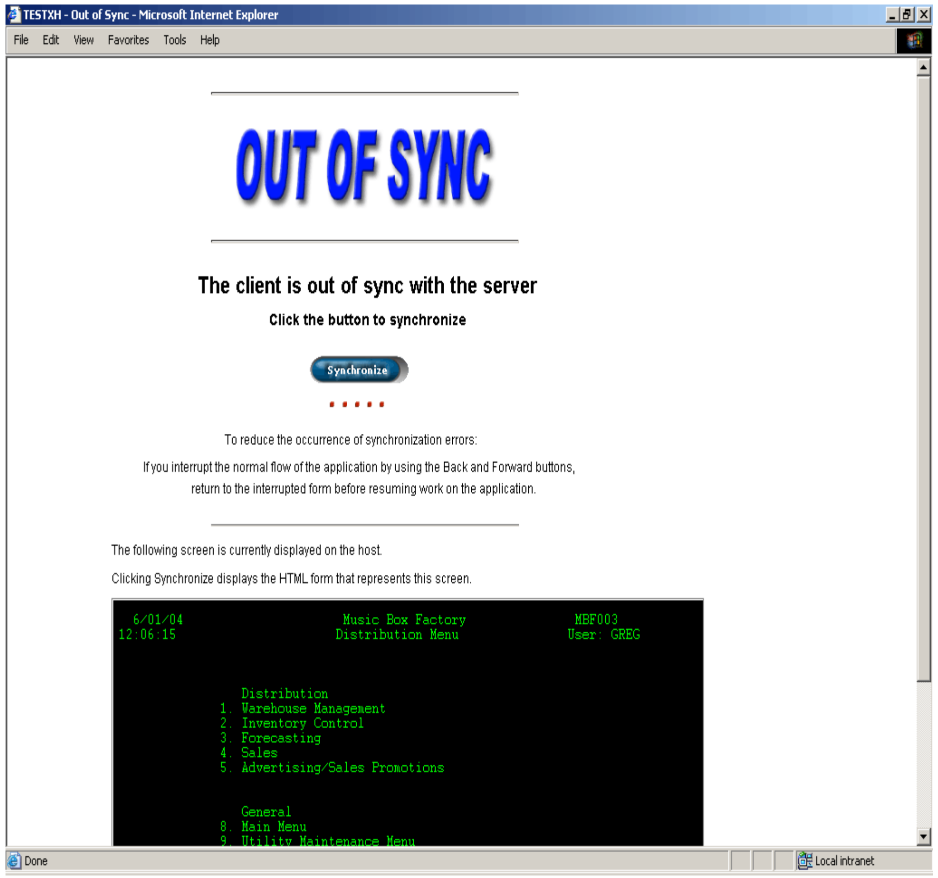
During Development

To incorporate a user extension into your Application during development:

1. Place the HTML file in the following directory on the development machine:
   `<InstallDir>\JacadaFiles\classes\appls\<ApplName>\xhtml\templates\user`
2. Make sure that the file bears the correct name.
3. Run the Application.

The OutOfSync Screen

When the Client and the JIS Server lose synchronization, and the end-user submits a form to the server, the JIS Server returns the out-of-sync screen:


TESTXH - Out of Sync - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Synchronize
Done
Local intranet

OUT OF SYNC

The client is out of sync with the server

Click the button to synchronize

To reduce the occurrence of synchronization errors:

If you interrupt the normal flow of the application by using the Back and Forward buttons, return to the interrupted form before resuming work on the application.

The following screen is currently displayed on the host.

Clicking Synchronize displays the HTML form that represents this screen.

```
  6/01/04                        Music Box Factory                MBF003
 12:06:15                       Distribution Menu                User: GREG


                 Distribution
              1. Warehouse Management
              2. Inventory Control
              3. Forecasting
              4. Sales
              5. Advertising/Sales Promotions


                 General
              8. Main Menu
              9. Utility Maintenance Menu
```

Figure 59. The out-of-sync screen

Note: By default, the out-of-sync screen appears **without** showing the screen currently displayed on the host. To display the host screen on the out-of-sync screen, add `DisplayHostImageWhenOutOfSync=1` to the `[XHTML]` section of the jacadasv.ini file:

The out-of-sync screen informs you that the end-user is displaying a window which does not represent the current host screen. If the out-of-sync screen appears, to fix this situation, the end-user must click the **Synchronize** button. The JIS Server then returns the correct window, which represents the current host screen. All of the recently submitted data, however, is lost.

A common cause for this is if the end-user navigates using the browser's **Back** and **Forward** buttons and submits a form different to the one previously sent by the JIS Server.

Another possible cause is erratic host behavior. To reduce such occurrences, it is recommended to use the **`SetWaitForScreenState`** DoMethod.

To regain lost data:

1. Click the browser's **Back** button, until you get to the screen, in which the lost data is displayed.
2. Copy the lost data.
3. Click the browser's **Forward** button, until you return to the correct screen.
4. Insert the lost data.

Note: You must be careful to return back to the correct screen. Otherwise you will lose synchronization again.

Customizing the OutOfSync Screen

The out-of-sync screen is similar to any Subapplication XHTML, except that the out-of-sync screen does not represent a host screen.

The process of creating an HTML extension for the out-of-sync screen is the same as for a Subapplication XHTML.

You name the HTML extension `"OutOfSync.html"` and place the file in the following directory as you would place any other HTML extension:

`<InstallDir>/Classes/appls/<ApplName>/xhtml/`**`user`**`/`

To make use of a custom OutOfSync screen, the runtime INI setting `OutOfSync` should be set to `Page`.

Using JavaScript in HTML Extensions

A JavaScript is a small program designed to run inside your XHTML, and automate some task, or to enhance a feature of your XHTML page. A JavaScript program consists of one or more instructions included with the HTML tags that form your *user* HTML extension files. When a browser encounters a JavaScript instruction, it stops to process it. For example, the instruction might tell the browser to format and display text and graphics on the page. JavaScript instructions are mixed together with the familiar HTML markup tags. You can use JavaScripts for various purposes, such as validating forms, enabling accelerator support, etc.

To incorporate a JavaScript into a *user* HTML extension file:

1. Create a `<Script>` tag in the user HTML's `<Head>`.
2. Create an event handler in the user HTML's `<Body>`.

Creating a <Script> Tag in the HTML's <Head>

There are two ways of creating a `<Script>` tag in the user HTML's `<Head>`:

- Creating a `<Script>` tag which contains the JavaScript.
- Creating a `<Script>` tag with a reference to an external JavaScript file.

Note: If your JavaScript code consists of special XML character, such as "<", ">", and "&", then you must create the <Script> tag with a reference to an external JavaScript file.

To create a `<Script>` tag which contains the JavaScript:

1. Create the opening and closing `<Script>` tags in the HTML's `<Head>`.
2. Insert the JavaScript between the opening and closing tags.

To create a `<Script>` tag with a reference to an external JavaScript file:

1. Save your JavaScript as a separate `*.js` file.
2. Create the opening and closing `<Script>` tags in the HTML's `<Head>`. Make sure there is a space between the opening and closing tags.
3. Inside the opening tag insert an `"src="` attribute with a reference to the **`*.js`** file. You must provide the full URL path to the `*.js` file, so that the Client can access it via the web server.

Example 6. Link to a JavaScript file in a <script> tag

This example shows how to create a <script> tag which points to an external Javascript file.

```
<Script type="text/javascript" src="http://190.40.17.770/appls/Xhtml01/
xhtml/user/script.js">
```

Creating an Event Handler in the HTML's <Body>

You insert an event handler either as an attribute of the `<Body>` opening tag itself or as an attribute of a tag within the HTML's `<Body>`, such as `<Input>`, `<Button>` or `<IMG>`.

You may use the following event handlers to invoke a JavaScript function:

- `onClick`

- `onDblClick`
- `onMouseDown`
- `onMouseMove`
- `onMouseUp`
- `onMouseOver`
- `onMouseOut`
- `onLoad`
- `onUnload`
- `onFocus`
- `onBlur`
- `onKeyDown`
- `onKeyPress`
- `onKeyUp`
- `onReset`
- `onSelect`
- `onChange`

Example 7. An onClick event handler

In the following example, a button with an `onClick` event handler initiates a JavaScript function named “JFunction”:

```
<Input type="button" name="Validate" value="Submit"
   onClick="JFunction()">
```

Examples of User HTML Extensions

This section provides several examples of the use of user HTML extensions.

The following examples are provided:

- Changing the title of all HTML pages to your company’s name
- Setting a different background image for each Subapplication
- Inserting a JavaScript-driven animation into your Application
- Enabling FKey support

Changing the Title of all HTML Pages to Your Company's Name

The following user HTML extension file is named `appl.html` and placed under the JIS Server's *user* directory. The `<Title>` tag in this file overrides the *original* JIS-generated XHTMLs' `<Title>` tag on an Application level.

```
<Html>

   <Head>
      <Title>Orange Planet Industries</Title>
   </Head>

</Html>
```

Setting a Different Background Image for Each Subapplication

For three Subapplications you create the following three HTML files:

```
<html>
   <body bgcolor="blue"/>
</html>

<html>
   <body bgcolor="green"/>
</html>

<html>
   <body bgcolor="yellow"/>
</html>
```

You must give each of these three files the name of their respective Subapplication XHTMLs and place them in the *user* directory.

Inserting a JavaScript-Driven Animation into Your Application

The following user HTML extension file contains a JavaScript, which calls upon a series of 10 GIF files named `Animation1.gif`, `Animation2.gif`, `Animation3.gif`, etc. These GIF files are repeatedly displayed one after the other in set intervals, forming an animation effect. To use GIF files in the extension, the GIF files must be stored under the JIS Server's *images* directory.

The location of the GIF files is given as a URL address in the `SRC` attribute, as they are accessed through the web server.

```
<HTML>

<BODY BGCOLOR="aqua">

<IMG NAME="Animation" SRC="http://226.19.18.120/classes/appls/<ApplName>/
images/Animation1.gif"/>

<SCRIPT LANGUAGE="JavaScript">

var i,d;

i=0;
```

```
function anime() {
   d=i%10+1;
   document.images.Animation.src='http://226.19.18.120/classes/
   appls/<ApplName>/images/Animation'+d +'.gif';
   setTimeout("anime()",1000);
   i=i+1;
   }
anime();
</SCRIPT>
</BODY>
</HTML>
```

Enabling FKey Support

There are two methods of enabling FKey support for your application.

In Version 9.0

As of version 9.0, webMethods JIS has simplified the process of enabling FKey support in the XHTML client.

To enable support for FKeys:

Set the parameter `FkeySupport` to 1. The `FkeySupport` parameter is in the `XHTML` section of the `<ApplName>.ini` file.

```
[XHTML]
FkeySupport = 1
```

Note: The FkeySupport runtime ini parameter enables Fkeys only, but not other accelerators such as CTRL+C, PAGE-UP, and so on.

For All Versions

In all versions of webMethods JIS, FKey support can be implemented via an HTML extension.

In this example we implement FKey support through an HTML extension to your JITGUI Subapplication. You must name this file `jitgui.html` and place it in your `<InstallDir>\appls\<ApplName>\xhtml\user\` directory.

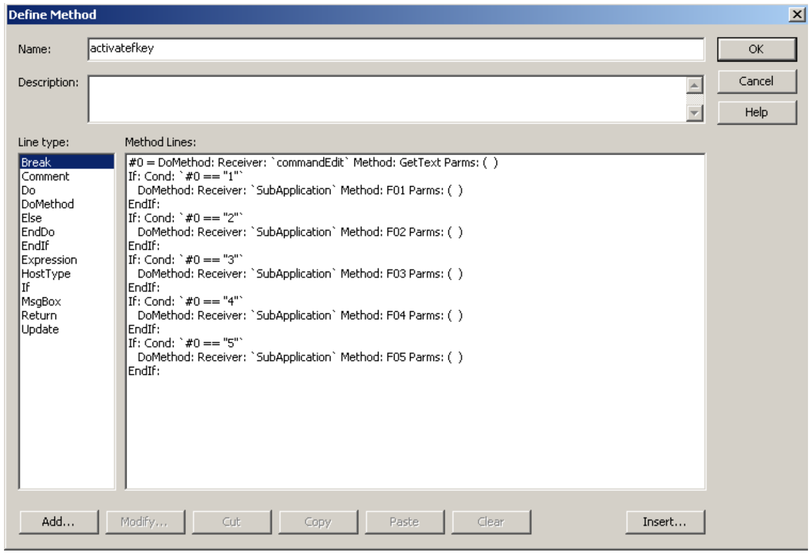
Add the following controls to the JITGUI Subapplication in ACE:

- A hidden edit field, named **commandEdit**.
  This edit is going to contain the command executed.
- A hidden button, named **fbutton**.
  This button is going to simulate all FKeys.

Both the edit field and the button are hidden using the HideControl event in the ***UserInitSubApplication*** System-Triggered method. Note that both need to have their Runtime Data Flow set in both directions, for the HideControl to work.

The process invoked by this extension, captures the FKey, writes a designated string per fkey in the hidden edit, and then simulates a pressing of the button.

In ACE, a method attached to the new hidden button checks the string in the hidden text box, and activates a specific fkey function according to that string.



Name: activatefkey

```
#0 = DoMethod: Receiver: `commandEdit` Method: GetText Parms: ( )
If: Cond: `#0 == "1"`
  DoMethod: Receiver: `SubApplication` Method: F01 Parms: ( )
EndIf:
If: Cond: `#0 == "2"`
  DoMethod: Receiver: `SubApplication` Method: F02 Parms: ( )
EndIf:
If: Cond: `#0 == "3"`
  DoMethod: Receiver: `SubApplication` Method: F03 Parms: ( )
EndIf:
If: Cond: `#0 == "4"`
  DoMethod: Receiver: `SubApplication` Method: F04 Parms: ( )
EndIf:
If: Cond: `#0 == "5"`
  DoMethod: Receiver: `SubApplication` Method: F05 Parms: ( )
EndIf:
```

Figure 60. The hidden button’s method

```
<html><head><script>
   if (!document.all) {
      document.captureEvents(Event.KEYDOWN);
   }
   document.onkeydown =writeAndSubmit;
   function writeAndSubmit(event){
      var key;
      if (document.all) {
         key = window.event.keyCode;
      }
      else {
         key = event.keyCode;
      }
```

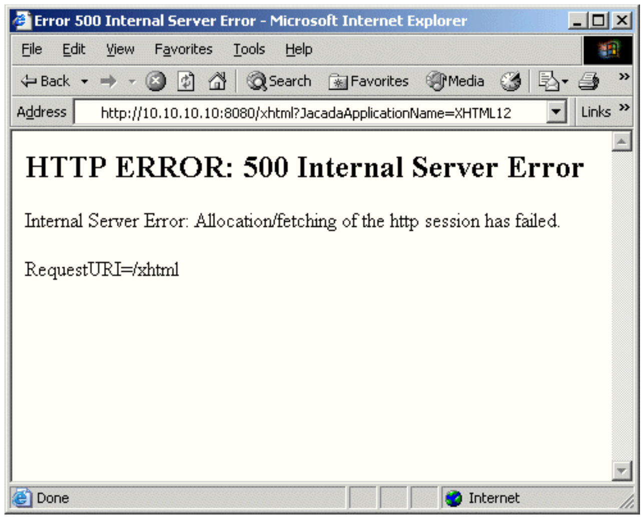
```
        var val = 0;
        if (key ==112) {
           val = "1";
        }
        else if (key ==113) {
           val = "2";
        }
        else if (key ==114) {
           val = "3";
        }
        else if (key ==115) {
           val = "4";
        }
        else if (key ==116) {
           val = "5";
        }
        else return true;
        document.jacadaform.elements["commandEdit"].setAttribute("value",val);
        document.jacadaform.elements["btn_fbutton"].click();
        if (document.all) {
           window.event.keyCode = 0;
           window.event.returnValue = false;
           window.event.cancelBubble = true;
        }
        else {
           event.keyCode = 0;
           event.returnValue = false;
           event.cancelBubble = true;
        }
     }
  </script></head></html>
```

Note: This script cancels the default browser handling of the F1 key and works on both Netscape and Internet Explorer.

Customizing HTML Error Pages

The JIS Server responds to each client request by returning a status code. If the status code indicates an error, then the client displays an error message.

The “HTTP 500 - Internal Server Error” message, seen below, is typical of standard error pages that leave many end-users at a loss as to how to proceed:


Error 500 Internal Server Error - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://10.10.10.10:8080/xhtml?JacadaApplicationName=XHTML12
Links
Done
Internet

HTTP ERROR: 500 Internal Server Error

Internal Server Error: Allocation/fetching of the http session has failed.

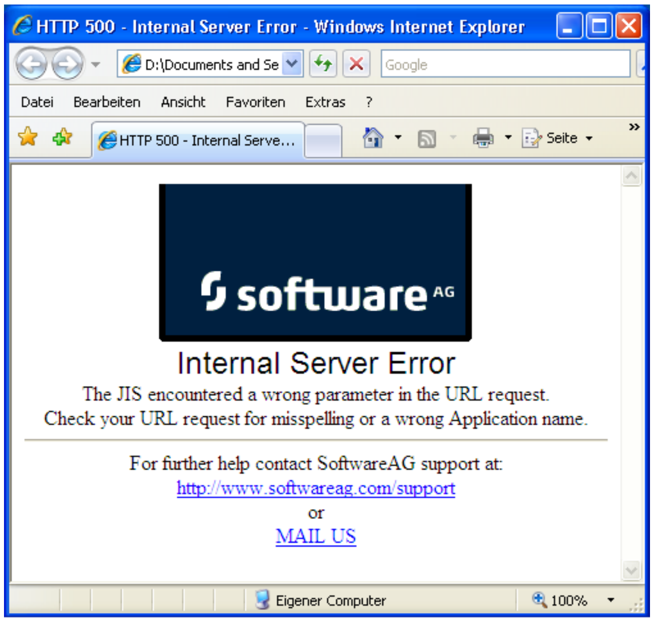
RequestURI=/xhtml

Figure 61. Standard HTTP error message

Note: The response is browser-dependent and subject to each browser’s settings.

With little effort, you can replace these standard error pages with customized error pages that inform your end-users of the error, indicate what may have caused this error, and how to deal with the error, as well as provide useful links.

This way, the same “HTTP 500 - Internal Server Error” message, seen previously, may be displayed like this:


HTTP 500 - Internal Server Error - Windows Internet Explorer
D:\Documents and Se
Google
Datei Bearbeiten Ansicht Favoriten Extras ?
HTTP 500 - Internal Serve...
Seite
software AG
Eigener Computer
100%

Internal Server Error

The JIS encountered a wrong parameter in the URL request.
Check your URL request for misspelling or a wrong Application name.

For further help contact SoftwareAG support at:
http://www.softwareag.com/support
or
MAIL US

Figure 62. Customized HTTP error message

Note: The settings for this feature are configured in the *`jacadasv.ini`* file.

To customize HTML Error Pages, you must:

- Write the replacement HTML Error Pages.
- Name the replacement HTML Error Pages correctly.
- Save the replacement HTML Error pages.

Error Page Configuration

In the `jacadasv.ini` file on the JIS Server, you can define the settings of the External Error Pages feature. This is done in the INI file's `[HTTP]` section.

You can set the following parameters:

- `UseExternalErrorPages=`
- `ExternalErrorPagesDir=`

Table 62. jacadasv.ini: error page parameters

Parameter	Explanation
`UseExternalErrorPages`	Specifies whether or not to enable customized HTML Error Pages. 1 - External Error Pages are enabled 0 - External Error Pages are disabled Default value is 1.
`ExternalErrorPagesDir`	Defines the directory, in which HTML Error Pages are placed. By default, this parameter is set to: `<InstallDir>\classes\HttpErrors`

Note: By default, the feature is enabled. You do not have to change the INI file settings, to use customized Error Pages.

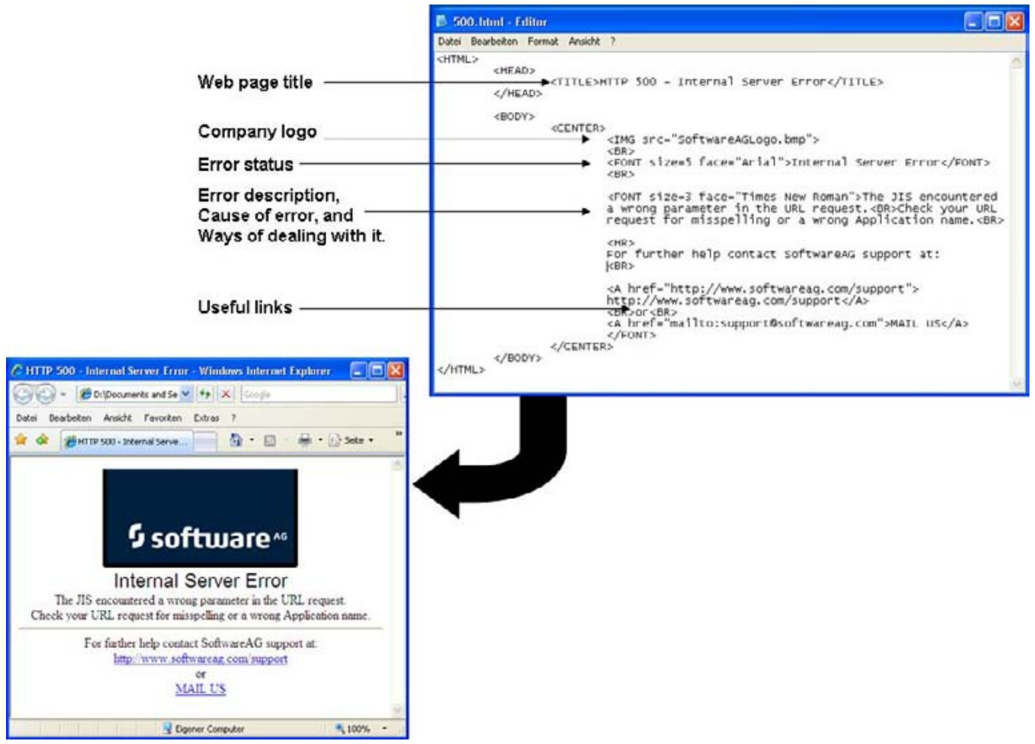
Writing HTML Error Pages

Any valid HTML page is acceptable. This page should contain an indication of the error type it represents. You can provide the cause of this error, as specifically pertaining to your Application, as well as a procedure for dealing with this error. You may also provide links to other pages, such as a troubleshooting page.

Note: It is possible to provide an HTML Error Page with a link that initiates a new host session. For this, you must provide the correct URL entry for invoking the webMethods JIS runtime.

Example 8. HTML code for error page

This example shows the HTML code for the previous error page:


Web page title
Company logo
Error status
Error description, Cause of error, and Ways of dealing with it.
Useful links
500.html - Editor
Datei Bearbeiten Format Ansicht ?
HTTP 500 - Internal Server Error - Windows Internet Explorer
Datei Bearbeiten Ansicht Favoriten Extras ?
software AG
Eigener Computer
100%

```
<HTML>
        <HEAD>
                <TITLE>HTTP 500 - Internal Server Error</TITLE>
        </HEAD>
        <BODY>
                <CENTER>
                        <IMG src="SoftwareAGLogo.bmp">
                        <BR>
                        <FONT size=5 face="Arial">Internal Server Error</FONT>
                        <BR>
                        <FONT size=3 face="Times New Roman">The JIS encountered
                        a wrong parameter in the URL request.<BR>Check your URL
                        request for misspelling or a wrong Application name.<BR>
                        <HR>
                        For further help contact SoftwareAG support at:
                        <BR>
                        <A href="http://www.softwareag.com/support">
                        http://www.softwareag.com/support</A>
                        <BR>or<BR>
                        <A href="mailto:support@softwareag.com">MAIL US</A>
                        </FONT>
                </CENTER>
        </BODY>
</HTML>
```

Internal Server Error
The JIS encountered a wrong parameter in the URL request.
Check your URL request for misspelling or a wrong Application name.

For further help contact SoftwareAG support at:
http://www.softwareag.com/support
or
MAIL US

This error page informs of the error type, indicates what may have caused this error and how the end-user can deal with it. Also provided are two links.

Including Original Error Text

When you write new error pages you must write the entire text that is to appear on the page. You can, however, utilize the error message text from the original error page by adding the string `$jacadaerror$` to the HTML code.

In runtime, when the error page appears, `$jacadaerror$` is replaced with the text from the original error message.

Naming HTML Error Pages

Each error status has its own code. By convention you name an HTML Error Page file `<ErrorCode>.html`, according to the error code it represents. The customized error page seen on the previous page would, therefore, be named `500.html`.

All possible error codes are listed at the end of this section.

There are two error types for which you can customize HTML Error Pages:

- Client-side errors
- Server-side errors

Client-side Errors

The 4xx class of status code is intended for cases in which the client seems to have erred. Except when responding to a HEAD request, the server **should** include an entity containing an explanation of the error situation, and whether it is a temporary or permanent condition. These status codes are applicable to any request method. User agents **should** display any included entity to the user.

Server-side Errors

Response status codes beginning with the digit "5" indicate cases in which the server is aware that it has erred or is incapable of performing the request. Except when responding to a HEAD request, the server **should** include an entity containing an explanation of the error situation, and whether it is a temporary or permanent condition. User agents **should** display any included entity to the user. These response codes are applicable to any request method.

Saving HTML Error Pages

Save the HTML Error Pages into the correct directory.

This directory is specified in the `jacadasv.ini` file's `[HTTP]` section, as the value for the `ExternalErrorPagesDir` parameter.

If for example the INI file code reads:

```
ExternalErrorPagesDir=C:\JacadaRuntime\classes\HttpErrors
```

Then save the file under `C:\JacadaRuntime\classes\HttpErrors\`.

Error Code Explanations

The following tables list and explain all error codes and their respective HTML file names. This list is for your convenience and does not imply that you need to create a replacement for every error type.

For further explanations, you can also see the official W3C website at:

http://www.w3.org/protocols/rfc2616/rfc2616-sec10.html

Table 63. HTML error codes (Sheet 1 of 4)

Error Code	Error Status	Explanation	File Name
Client-side			
400	Bad Request	The client request was not understood by the server due to malformed syntax.	400.html
401	Unauthorized	The URL request requires user authentication. The client may repeat the request with a suitable Authorization header field.	401.html
402	Payment Required	Code is reserved for future use.	402.html
403	Forbidden	The server understood the client request, but is refusing to fulfill it.	403.html
404	Not Found	The server has not found anything matching the client Request.	404.html
405	Method Not Allowed	The method specified in this URL is not allowed for the resource identified by the request.	405.html
406	Not Acceptable	The resource identified by the request is only capable of generating response entities which have content characteristics not acceptable according to the Accept headers sent in the request.	406.html

Table 63. HTML error codes (Sheet 2 of 4)

Error Code	Error Status	Explanation	File Name
`407`	Proxy Authentication Required	This code indicates that the client must first authenticate itself with the proxy. The client may repeat the request with a suitable Proxy-Authorization header field.	`407.html`
`408`	Request Timeout	The client did not produce a request within the time that the server was prepared to wait.	`408.html`
`409`	Conflict	The request could not be completed due to a conflict with the current state of the resource.	`409.html`
`410`	Gone	The requested resource is no longer available at the server and no forwarding address is known.	`410.html`
`411`	Length Required	The server refuses to accept the request without a defined Content- Length.	`411.html`
`412`	Precondition Failed	The precondition given in one or more of the request-header fields evaluated to false when it was tested on the server.	`412.html`
`413`	Request Entity Too Large	The server is refusing to process a request because the request entity is larger than the server is willing or able to process.	`413.html`
`414`	Request-URI Too Long	The server is refusing to service the request because the request-URI is longer than the server is willing to interpret.	`414.html`

Table 63. HTML error codes (Sheet 3 of 4)

Error Code	Error Status	Explanation	File Name
`415`	Unsupported Media Type	The server is refusing to service the request because the entity of the request is in a format not supported by the requested resource for the requested method.	`415.html`
`416`	Request Range Not Satisfiable	The client request included a Range request-header field, and none of the range-specifier values in this field overlap the current extent of the selected resource, and the request did not include an If-Range request-header field.	`416.html`
`417`	Expectation Failed	The server could not meet the expectation given in an Expect request-header field.	`417.html`
Server-side			
`500`	Internal Server Error	The server encountered an unexpected condition which prevented it from fulfilling this request.	`500.html`
`501`	Not Implemented	The server does not support the functionality required to fulfill the client request.	`501.html`
`502`	Bad Gateway	While acting as a gateway or proxy, the server received an invalid response from the upstream server it accessed in attempting to fulfill the request.	`502.html`

Table 63. HTML error codes (Sheet 4 of 4)

Error Code	Error Status	Explanation	File Name
`503`	Service Unavailable	The server is unable to handle the request due to a temporary overloading or maintenance of the server.	`503.html`
`504`	Gateway Timeout	While acting as a gateway or proxy, the server did not receive a timely response from the auxiliary server it needed to access in attempting to complete the request.	`504.html`
`505`	HTTP Version Not Supported	The server does not support, or refuses to support, the HTTP protocol version used in this request.	`505.html`

CSS Usage in the JIS XHTML Client

webMethods JIS generates CSS files to reduce the size of the application's XHTML pages. Customers can take advantage of this feature to apply their own CSS style sheets to their application windows. You can use CSS to create a style sheet that supplies a generic layout to several pages, creating a uniform look for all of the pages.

CSS – A Basic Explanation

Cascading Style Sheets, or CSS, is a simple style sheet mechanism that allows authors and readers to attach style (e.g. fonts, colors and spacing) to HTML documents. CSS lets you separate the content of a page from its appearance. The CSS language is human-readable and -writable, and expresses style in common desktop publishing terminology.

Style Information Levels

Style information for an HTML page can be supplied at different levels:

1 Inside the HTML element, called *inline* style declaration.

2 Inside the `<head>` tag; also called an *embedded* or *internal* style declaration.
3 In an external style sheet
4 By browser default settings.

The style information levels listed above are in order of priority, with inline style having the highest priority and browser defaults having the lowest priority.This means that a style declared inline overrides any similar style element defined inside the `<head>` tag, or in an external style sheet, or as a browser default.

Example 9. CSS and style information priorities

We have an external CSS (named `example.css`) containing the following rule:

```
h1, h2 {color: blue; font-size: 20}
/* selects all elements with tag <h1> or <h2>*/
```

And our page looks like this:

```
<head>
    <link REL="stylesheet" TYPE="text/css" href="example.css">
<style>
       h2 {color:red}
       h3 {font-size: 10}
   </style>
</head>
<body>
      <h1>This is header 1</h1>
      <h2>This is header 2</h2>
      <h3 style=”font-size:15”>This is header 3</h3>
</body>
```

Results:

The “h1” line will be blue with font-size 20.

The “h2” line will be red with font-size 20.

The “h3” line will get the color from the browser’s default, and its font size will be 15.

The CSS File in JIS

At runtime generation, CSS files are created, one for Microsoft Internet Explorer browsers and one for Netscape browsers. The files are called `kb_IE.css` and `kb_NETSCAPE.css.`

These files contain rules with class selectors, corresponding to the KnowledgeBase definitions used in the application. These files are recreated at every runtime generation. Example 10 shows an example of the contents of a JIS-generated CSS file.

The generated CSS files should not be modified. If you want to apply custom CSS to your screens, see “Using Your Own Style Sheet” on page 275.

The CSS files are created in:
`<InstallDir>\JacadaFiles\classes\appls\<ApplName>\xhtml\CSS`

Example 10. Contents of a JIS CSS file

```
.btn2 {          /* .WebLook_Border.Button */
font-family:”System”;
font-size:10px;
background-position:center top;
font-weight:normal;
width:1020;
background-repeat:no-repeat;
border-width:0;
font-style:normal;
background-color:transparent;
text-decoration:none;
background-image:url(/classes/appls/TEST03/images/weblookborder_1024.jpg);
}

.btn1 {          /* JITGDynamicFKey.Button_MenuItem_
Accelerator_GeneralUTMethod.Button */
font-family:”MS Sans Serif”;
font-size:11px;
color:#000080;
font-weight:bold;
border-width:0;
font-style:normal;
background-color:#ffffff;
text-decoration:underline;
}

.txt1 {          /* JITGInputFieldNumeric.TextBox.TextBox */
font-family:”Courier New”;
font-size:13px;
border-style:inset;
font-weight:normal;
text-align:right;
border-width:2;
font-style:normal;
text-transform:none;
text-decoration:none;
}

.lbl1 {          /* JITGDynamicOutputFieldBright.Static.Static */
font-family:”Courier New”;
font-size:15px;
font-weight:bold;
text-align:left;
overflow:hidden;
font-style:normal;
text-decoration:none;
}

.date1 {          /* .Date.Date */
font-family:”MS Sans Serif”;
font-size:11px;
color:WindowText;
font-weight:normal;
text-align:left;
font-style:normal;
background-color:#ffffff;
text-decoration:none;
}
```

The CSS file consists of rule classes containing descriptions of the properties of the controls used in the application. For each control used in the subapplication screens and defined in the KnowledgeBase, a class is created in the CSS file. If a particular control definition is used multiple times in the application, it is still represented by just one class in the CSS file.

The name of each class is formed according to the rules shown in Table 64, where each general type of control is assigned a specific abbreviation. For example, classes related to buttons have names that begin with `btn`. The names are completed by the addition of a numeric suffix. For example, if the application uses a `Frame` control and a `GroupBox` control, since each of these controls exist as separate entities in the KnowledgeBase, a separate rule class for each appears in the CSS file.

Table 64. Prefixes of CSS class names

Control Type	Abbr.	Control Type	Abbr.
Button	btn	Link	lnk
Check box	chk	Prompt	prm
Combo box	cbo	RadioGroup	rdg
Date	date	Spin	spn
Edit box Control	txt	Static	lbl
Frame	fra	SubWindow	swn
GroupBox	fra	Table	tbl
Line	lin	Text box	txt

According to Table 64, a `Frame` and a `GroupBox` both receive the CSS class name `fra`. To distinguish one from the other, a numeric suffix is assigned to all rule classes. So the class rules for a `Frame` and a `GroupBox` could be named `fra01` and `fra02`. The description beside the name helps you determine the specific button definition being described by the class.

The CSS Modifies the XHTML Pages

Example 11 shows a much simplified example of an XHTML page that was created at runtime generation. It represents a single subapplication screen. The bolded elements are related to the CSS in Example 10, also created at runtime generation.

Note the link command:

```
<link rel="stylesheet" type="text/css" href="/classes/appls/kb_IE.css"/>.
```

It was added automatically to the generated XHTML page. This command points to the CSS style sheet that was automatically created for this screen.

Further down, in several places the class keyword appears, for example:

```
class="lbl1"
```

Each use of the class keyword is a reference to specific a rule class in the CSS file pointed to by the `link` command. You can see the rule classes in Example 10. The properties defined in CSS class `lbl1`, for example, apply to the Static002 object in this example. The properties defined in the CSS class override any similar, implicit properties in the XHTML template. However, all properties explicitly stated in the XHTML template have priority (unless the **`important`** keyword is used in the CSS definition, which will not be the case with respect to the generated CSS files). Refer again to Example 9 if this is not clear.

Example 11. XHTML template referring to CSS

This is an example of an XHTML template with referral to an external CSS style sheet:

```
<?xml version=”1.0” encoding=”UTF-8”?>
<html lang=”en” xmlns=”http://www.w3.org/1999/xhtml”>
<head xmlns=””>
<meta http-equiv=”EXPIRES” content=”0”/>
<meta http-equiv=”Pragma” content=”no-cache”/>
<meta http-equiv=”Content-Type” content=”text/html; charset=utf-8”/>
<title>TEST03 - Main Menu</title>
<script language=”JavaScript1.2” type=”text/javascript” src=”/classes/js/
jacada.js”> </script>
<link rel=”stylesheet” type=”text/css” href=”/classes/appls/kb_IE.css”/>
</head>
<body tabindex=”-1” onload=”setFocusedControl(&quot;Sub1103201006&quot;);
displayMessageBox(); setTitle(); formatLoadedDates(); writeSessionId();”
bgcolor=”#ffffff” xmlns=””>
<form name=”jacadaform” id=”jacadaform” action=”http://” method=”post”
onsubmit=”formatSubmitedDates();”>
<pre style=”position: absolute; top: 55; left: 18; height: 21; width: 240;
overflow: hidden;”>
<label id=”Static002” class=”lbl1” style=”position: absolute; left: 0; top:
0; width: 240; height: 21;
overflow: hidden; color: ButtonText; background-color: #ffffff; font-family:
Arial Black; font-size: 16px; font-weight: bold; font-style: normal; text-
decoration: none;
text-align: left; line-height: 21px;”>Enter account number</label>
</pre>
<pre style=”position: absolute; top: 74; left: 15; height: 22; width: 277;
overflow: hidden;”>
<label id=”Static” class=”lbl1” for=”Sub1103201006” style=”position:
absolute; left: 0; top: 0; width: 277; height: 22; overflow: hidden; color:
#000000; background-color: #ffffff; font-family: Arial Black; font-size:
16px; font-weight: bold; font-style: normal; text-decoration: none; text-
align: left; line-height: 22px;”>and select desired option</label>
```

```
</pre>
<input type=”text” name=”date_Date” id=”Date” class=”date1” value=””
tabindex=”8” title=”” onfocus=”saveFocusedControl(&quot;Date&quot;);”
style=”font-family: MS Sans Serif; font-size: 11px; font-weight: normal;
font-style: normal; text-decoration: none; text-align: left;
position:absolute; left: 0; top: 0; width: 80; height: 21; color:
WindowText; background-color: #ffffff;” maxlength=”10”
onchange=”validateDate(this.value, this)” onkeydown=”tabDate(this.value,
this)”/>
<img name=”date_Date” src=”i:\java\classes/appls/TEST03/images/_DATE7.gif”
tabindex=”8” style=”position: absolute; top: 0px; left: 77; height: 17;
width: 16px; border-style: outset; border-width: 2px; cursor: hand;”
onclick=”showDate(&quot;Date&quot;)”
onkeypress=”dateBtnPressed(&quot;Date&quot;)”/>
<input type=”submit” name=”btn_Sub0500501004” id=”Sub0500501004”
class=”btn1” value=”ADD” tabindex=”9” title=””
onfocus=”saveFocusedControl(&quot;Sub0500501004&quot;);” alt=”ADD”
style=”font-family: Arial Black; font-size: 13px; font-weight: bold; font-
style: normal; text-decoration: none; position:absolute; left: 59; top: 233;
 width: 103; height: 35; color: #000000; background-color: ButtonFace;
background-image: url(i:\java\classes/appls/TEST03/images/button1.gif);
background-repeat: no-repeat; background-position: center center; text-
align: center; cursor: hand;”/>
<input type=”submit” name=”btn_Sub0600501007” id=”Sub0600501007”
class=”btn2” value=”UPDATE” tabindex=”10” title=””
onfocus=”saveFocusedControl(&quot;Sub0600501007&quot;);” alt=”UPDATE”
style=”font-family: Arial Black; font-size: 13px; font-weight: bold; font-
style: normal; text-decoration: none; position:absolute; left: 60; top: 274;
width: 100; height: 35; color: #000000; background-color: ButtonFace;
background-image: url(i:\java\classes/appls/TEST03/images/button1.gif);
background-repeat: no-repeat; background-position: center center; text-
align: center; cursor: hand;”/>
<pre style=”position: absolute; top: 692px;”>
<label id=”DIL” style=”position: absolute; left: 0; top: 0;”>  </label>
</pre>
</form>
</body>
</html>
```

KBInformation.xml File Not to be Modified

Along with the CSS files created at runtime, the `KBInformation.xml` file is also created. It is created in the `<InstallDir>\appls\<LibName>` directory if you are using subapplication libraries; otherwise the CSS files are created in `<InstallDir>\appls\<ApplName> directory`. The `KBInformation.xml` file is used by webMethods JIS to ensure that the class names for the CSS rules in the CSS files stay the same from one runtime generation to another. Do not modify the `KBInformation.xml` file.

<ApplName>.ini File Setting

The use of CSS by the application is controlled by a runtime *.ini file setting (in file `<applname>.ini`) in the XHTML section of the file. The setting is `UseCSSDefinitions`; valid values are 1 (yes) and 0 (no). The default setting is 1.

```
[Xhtml]
UseCssDefinitions=1
```

Modifying the Generated CSS File

The generated CSS files should not be modified. If you want to apply custom CSS to your screens, see "Using Your Own Style Sheet" on page 275.

General Recommendations Regarding CSS

The following generic recommendations apply when using CSS:

- To keep your HTML pages as small as possible, modify the appearance of your subapplication screen appearance via the KnowledgeBase, rather than through CSS.
- A subapplication with no local modifications results in the smallest HTML page.
- Local modifications in a subapplication produce a larger page because those modifications now appear "inline" in the HTML page.
- Be aware that, when skins are used, if the skin sets a value for a control that is different from the value specified in the KnowledgeBase for that control, the skin values appear as inline modifications to the HTML page.
- Remember that if you can change the KnowledgeBase definition to reflect your desired changes, you eliminate the need to deal with it through CSS. This, indeed, is the preferred approach.

Java Skins vs. Custom CSS

If you want to modify the look-and-feel of your application without applying the changes permanently to your KnowledgeBase, you have at least two robust options: you can use Java skins (discussed in "Applying Skins Across Applications" beginning on page 290) or you can use custom CSS. Which of these two options is best?

From a purely programing-oriented perspective, skins are probably the easier solution. To use custom CSS it is necessary to use Java extensions anyway, to change the CSS rule class used by the XHTML controls or to remove inline properties, and you must add a link to your custom CSS file to the original XHTML template. So using Java extensions works out to be less work overall for the developer.

However, in a development environment where the web design function is separate from the programming function, custom CSS may be more convenient. The designer of a web site is usually familiar with colors, graphics, and HTML. He probably knows or can easily learn how to write a CSS file, but may not know how to write Java code. In such a case, it makes sense for the designer to write a CSS file and hand it off to the application developer, who will then do the additional work of applying the CSS file to the application through skins or through HTML extensions.

Using Your Own Style Sheet

You can use your own CSS file by referencing it from your HTML template, or by invoking it through a Java extension.

Adding a Custom CSS file via an HTML Template

Code your CSS rule classes and save them in a file with the `.css` suffix.

Edit the HTML template for the screen on which you want to apply your CSS style sheet.In the `head` section, add a statement like the following to link to your style sheet:

```
<link rel=”stylesheet” type=”text/css” href=”/mydirectory/myfile.css”/>
```

To use your style sheet along with the automatically generated style sheet, make sure that you place the link statement for your CSS file **after** the link statement for the generated CSS file, so that your CSS sheet will have precedence.

Software AG recommends you use the “link” command and not the “@import” command to invoke the CSS through the HTML template, to insure proper behavior.

For Advanced Users Only: Additional Methods for CSS

The methods described in this section modify the class `com.jacada.jis .runtime.server.frontend.xhtml.controls.XhtmlControl`. Only advanced users should make use of these methods.

Methods to Set the CSS Class for a Control

`public void setCSSClass (String className)`

Sets a CSS class to the control. The class attribute is added to the XML element that contains the style of the control. If the class already exists, the new class overrides the old one.

`public void setCSSClass (Collection classNames)`

Sets all the CSS class names in this collection to the control.

`public void addCSSClass (String className)`

Adds the class to the control without removing the old class definition.

`public void removeCSSClass ()`

Detaches the control from all classes.

`public void removeCSSClass (String className)`

Detaches the control from a specific class.

`public Collection getCSSClass ()`

Returns the class names that this control belongs to.

Methods to Remove Style Definitions

The following methods remove style definitions from the inline style attribute so that a CSS class rule can affect the control.

`public void removeInlineStyle ()`

Removes all inline style properties, with the exception of size (width, height) and location (top, left).

`public void removeInlineStyle (InlineStyleProperty property)`

Removes a specific property.

`public void removeInlineStyle (Collection properties)`

Removes a set of properties.

Invoking a Custom CSS file via a Java Extension

You can apply a custom, external CSS file to an HTML page in the XHTML client through the use of a specific method in a Java extension.

The method is `addExternalCSSFile` in class: `com.jacada.jis.runtime.server.frontend.xhtml.controls.Window`

Its syntax is: `public void addExternalCSSFile(String url)`

Use the `url` parameter to pass the URL of the CSS file. This URL can be a path that is relative to the root directory of the web server or any other valid URL.

Limitations in the Use of External CSS Style Sheets

The following limitations apply to the use of CSS in an file external to the application's HTML template.

Inline Property Definitions Have Priority

Be aware that any inline property definitions in the HTML template have precedence over CSS style sheet definitions, unless the **`important`** keyword is used in the CSS definition. Inline property definitions include the result of changes applied to KnowledgeBase objects locally (i.e., changes applied to the object in your application and not in the KnowledgeBase itself).

"Get" Methods Considerations

The "get" methods (of class XhtmlControl) that return style-related information (such as methods `getBorderColor`, `getFontStyle`, `getFontSize`) always return information from inline style declaration or from the system generated CSS file, even if custom CSS files of your own are in effect.

Chapter 9. Enhancing Your Application Using Java Extensions

After the conversion of an application, you may want to further enhance its look or functionality. This can be done using user code extensions. It is possible to write HTML extensions and also Java extensions. This chapter focuses on enhancing your application using Java extensions.

The following topics are discussed:

- "Events for Activating Java Extensions" on page 279
- "Page-Level Extensions" on page 281
- "Control-level Extensions - onControlReady" on page 289
- "Java Extension Examples" on page 293
- "Extensions to the Date Calendar Window" on page 314

JIS's Javadoc Files

webMethods JIS provides you with a set of html files that include explanations of how to work with the generated Java classes in your application, and the methods they contain.

The javadoc files relevant to the XHTML client are installed under the following directory: `<InstallDir>\JacadaFiles\docs\xhtml`

Note: Only public methods are documented in the javadoc.

Events for Activating Java Extensions

In order to activate Java extensions, a set of three events is provided. These events occur during the on-going processing between the host and the client, which takes place in the XHTML Processing Module on the JIS Server.

There are three events available. The first two events occur during the process of building the XHTML to send to the client, whereas the last event is activated after the client submits the form:

- **`onControlReady`** - This event enables you to write extensions that affect a particular control type across the entire Application.

- **`onPageLoad`** - This event enables you to control the contents and the look of the HTML page. It occurs after the page has been updated and before it is sent to the client.
- **`onPageSubmit`** - This event enables you to manipulate user data after the client submits the form and before any server processing operations take place.

The following diagram illustrates when the events take place within the XHTML Processing Module on the JIS Server:

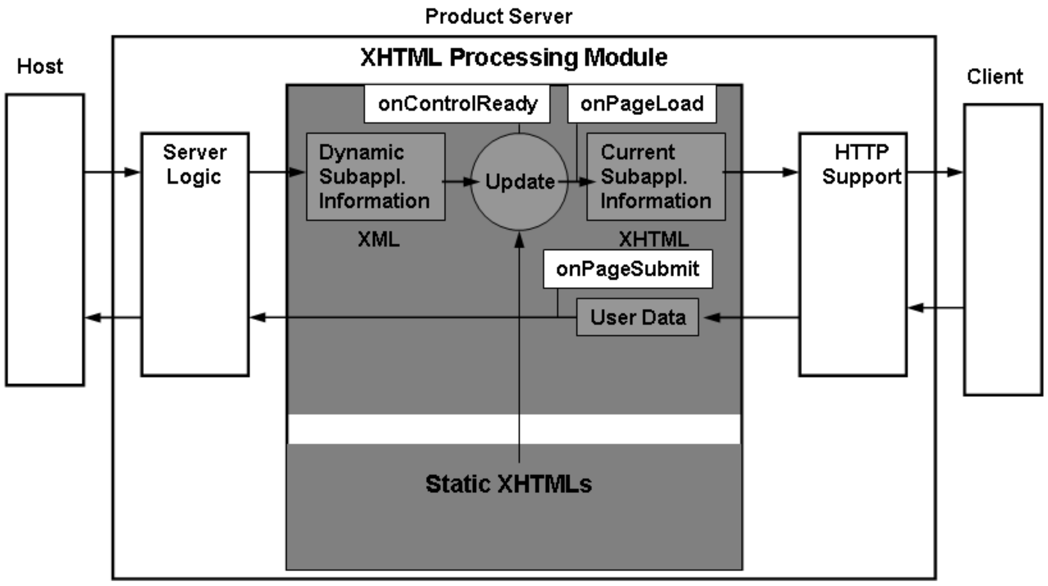


Figure 63. Events in the XHTML processing module

See Chapter 7 - "XHTML Runtime Architecture" on page 233 for more details on the XHTML Processing Module.

Extension Types

There are two main types of extensions that can be written:

- Page-level extensions
- Control-level extensions

To write page-level extensions, you use the **`onPageLoad`** and **`onPageSubmit`** events.

To write control-level extensions, you use **`onControlReady`** event.

The following sections describe in detail how to write extensions of each type.

Page-Level Extensions

In page-level extensions, you can manipulate the contents of the form either before the page is loaded onto the client browser, or manipulate user data immediately after the form is submitted by the client.

For this purpose, you use two events:

- **`onPageLoad`** - Use this event to manipulate the page before it is sent to the client browser.
- **`onPageSubmit`** - Use this event to manipulate user data after the client submits the form and before any server processing operations take place.

Extensions Activated Before Sending Page to Client - onPageLoad Event

The **`onPageLoad`** event occurs after the page has been updated and before it is sent to the client browser. This event enables you to control the contents and the look of the HTML page before it is displayed to the user.

The following extension capabilities are available:

- Adding and removing controls.
- Setting or retrieving control properties and values. For example, you can set/get control properties such as: size, location, text, fonts, colors, show/hide, enable/disable, balloon help, Z-order, etc.
- Adding JavaScript event handlers for a control. For example: `onClick`, `onMouseOver`, `onLoad`, `onFocus`, `onKeyDown`, etc.
- Adding JavaScript functions to be activated by one of the event handlers above. For example, a JavaScript activated by a button's `onClick` event.
- Retrieving data from a shared user variable and setting it in the page.
- Retrieving and setting cookies.

Extensions Activated After Submitting Page - onPageSubmit Event

The **`onPageSubmit`** event occurs immediately after the Client submits the page and before any server processing operations, such as formatting or applying dictionaries, are done on the user data.

The following extension capabilities are available:

- Changing post data before it is processed by the JIS Server and sent to the host.

- Storing data in a shared user variable.
- Creating and saving cookies.

This event can be used, for example, to manipulate the user data by converting data from user format to host format by removing masking. Another example is saving the submitted user data to an external database. Note, however, that you cannot use the **`onPageSubmit`** event to stop the submission of the page.

Extension Scope, Hierarchy and Files

The structure of a JIS Application is hierarchical. An Application may contain one or more Subapplications, and may also contain one or more libraries. A library contains one or more Subapplications. This hierarchy is at the base of the Java extensions mechanism.

Extension Scope

Page-level extensions are specific to an Application. Within an Application, the scope of an extension is determined by the user:

- Subapplication scope - affects a specific Subapplication only.
- Library scope - affects all the Subapplications in a specific library.
- Application scope - affects all the Subapplications in the entire Application.

Extension Hierarchy

Page-level extensions are activated in a hierarchical manner. For each Subapplication, first a Subapplication level extension is searched for. If it is found, the Subapplication level extension is activated. Otherwise, a library level extension is searched for. If it is found, the library level extension is activated. Otherwise, an Application level extension is searched for and if found, activated.

Additionally, Java inheritance exists between these three classes, where the Subapplication class inherits the library class, and the library class inherits the Applications class.

Extension Files

The **`onPageLoad`** code and the **`onPageSubmit`** code both reside in same file. All the user extension files reside in the following package:

```
appls.<ApplName>.xhtml.user
```

The name and location of the file depends on the scope of the extension.

Application Scope

For code that affects an entire application, the extension file must be named `Appl.java`.

The runtime generation process generates the following two files automatically:

- `Appl.java` - a skeleton file that enables you to add the user code.
- `jacc.bat` - a batch file for compiling the java code in this package.

Both these files are placed in the following Application source code directory:

```
<InstallDir>\jacadafiles\src\appls\<ApplName>\xhtml\user
```

After using `jacc.bat` to compile the Java source code, the classes are placed in the Application classes directory:

```
<InstallDir>\jacadafiles\classes\appls\<ApplName>\xhtml\user
```

For an application named APPL1, this is the code in the Appl.java file:

```
package appls.APPL1.xhtml.user;
import com.jacada.jis.runtime.server.frontend.xhtml.extension.*;
import com.jacada.jis.runtime.server.frontend.xhtml.context.*;
import com.jacada.jis.runtime.server.frontend.xhtml.controls.*;
/** description : Appl.java */
public class Appl implements
com.jacada.jis.runtime.server.frontend.xhtml.extension.IUserPageExtension {
   public Appl () {
   }
   public void onPageLoad(OnPageLoadContext context) {
   }
   public void onPageSubmit(OnPageSubmitContext onSubmitContext) {
   }
}
```

Library Scope

For code that affects all the Subapplications in a specific library, the extension file must be named `Appl.java`.

The runtime generation process generates the following two files automatically:

- `Appl.java` - a skeleton file that enables you to add the user code.
- `jacc.bat` - a batch file for compiling the java code in this package.

Both these files are placed in the following library source code directory:

```
<InstallDir>\jacadafiles\src\appls\<LibName>\xhtml\user
```

After using jacc.bat to compile the Java source code, the classes are placed in the library classes directory:

```
<InstallDir>\jacadafiles\classes\appls\<LibName>\xhtml\user
```

Both the Application extension file and the library extension file have the same name - `Appl.java`. The difference between the files is their location, and in the fact that the library extension code extends the Application extension code.

Example 12. Library-scope extension

For a library named LIB1, in an Application named APPL1, this is the code in the skeleton `Appl.java` file:

```
package appls.LIB1.xhtml.user;
import com.jacada.jis.runtime.server.frontend.xhtml.extension.*;
import com.jacada.jis.runtime.server.frontend.xhtml.context.*;
import com.jacada.jis.runtime.server.frontend.xhtml.controls.*;
/** description : Appl.java */
public class Appl extends appls.APPL1.xhtml.user.Appl {
   public Appl () {
      super();
   }
   public void onPageLoad (OnPageLoadContext context) {
      super.onPageLoad(context);
   }
   public void onPageSubmit (OnPageSubmitContext onSubmitContext) {
      super.onPageSubmit(onSubmitContext);
   }
}
```

Subapplication Scope

For code that affects a specific Subapplication only, the extension file must be named according to the Subapplication name: `<SubApplName>.java`.

For example, for a Subapplication named `SUBAPP1`, the name of the file is `SUBAPP1.java`.

In contrast to the skeleton files created automatically for Application and library extensions, a Subapplication extension file must be created by the user. The location and contents of the Subapplication file depend on whether the Subapplication resides in the main Application or in a library.

Subapplications Residing in a Library When a Subapplication resides in a library, the `<SubApplName>.java` file must be placed in the library source code directory, together with the library's `Appl.java` file:
`<InstallDir>\jacadafiles\`**`src`**`\appls\<`**`LibName`**`>\xhtml\user`

After using `jacc.bat` to compile the Java source code, the classes are placed in the library classes directory:
`<InstallDir>\jacadafiles\`**`classes`**`\appls\<`**`LibName`**`>\xhtml\user`

The Subapplication class should extend the library class. In such a case, the **`onPageLoad`** and **`onPageSubmit`** methods may call their super methods, as defined in the library extension.

Subapplications Residing in an Application When a Subapplication resides in an Application, the `<SubApplName>.java` file must be placed in the Application source code directory, together with the Applications's `Appl.java` file:
`<InstallDir>\jacadafiles\`**`src`**`\appls\<`**`ApplName`**`>\xhtml\user`

After using `jacc.bat` to compile the Java source code, the classes are placed in the Application classes directory:
`<InstallDir>\jacadafiles\`**`classes`**`\appls\<`**`ApplName`**`>\xhtml\user`

The Subapplication class should extend the Application class. In such a case, the **`onPageLoad`** and **`onPageSubmit`** methods may call their super methods, as defined in the Application extension.

Example 13. Subapplication-scope extension

Subapplication SA1 is in Application APPL1. To create a Subapplication scope extension, create a file named `SA1.java` and use the skeleton below:

```
package appls.APPL1.xhtml.user;
import com.jacada.jis.runtime.server.frontend.xhtml.extension.*;
import com.jacada.jis.runtime.server.frontend.xhtml.context.*;
import com.jacada.jis.runtime.server.frontend.xhtml.controls.*;
public class SA1 extends Appl {
   public void onPageLoad(OnPageLoadContext context) {
      ... // Subapplication specific code
      super.onPageLoad(context);
   }
   public void onPageSubmit(OnPageSubmitContext onSubmitContext) {
      ... // Subapplication specific code
      super.onPageSubmit(onSubmitContext);
   }
}
```

Note: When the Subapplication resides in a library, all you need to do is replace the Application name in the first line with the library name.

This example demonstrates how extensions are activated in runtime.

- Application APPL1 contains a library named LIB1.
- LIB1 contains two Subapplications - SA1 and SA2.
    - APPL1 has an Application scope extension.
    - LIB1 has a library scope extension.
    - SA1 has a Subapplication scope extension.
    - SA2 does not have a Subapplication scope extension.

In runtime, after SA1 is ready, its Subapplication extension code is activated. This code may call the library scope extension. The library extension code may call the Application scope extension.

After SA2 is ready, the LIB1 library extension code is activated. This code may call the APPL1 Application scope extension.

Note: If a Subapplication scope extension exists only for one event, such as `onPageLoad` for example, then for the `onPageSubmit` event the library or Application extension will be executed, depending on the implementation.

Java API - onPageLoad, onPageSubmit

A Java API is provided to enable you to write extensions. The full details of the Java API are described in a Javadoc that is provided with the product. The Javadoc can be found at `<InstallDir>\JacadaFiles\docs\xhtml`. This section, however, reviews a few highlights of the following packages included in the API:

- `com.jacada.jis.runtime.server.frontend.xhtml.controls`
- `com.jacada.jis.runtime.server.frontend.xhtml.context`

Controls Package

The controls package defines the control classes to be used in extensions.

These classes can be divided into the following categories:

- Simple Controls
- Container Controls
- Table Controls

Simple Controls

- Button
- Check box
- Combo box
- Date
- Line
- Link
- Multiline Text box
- Prompt
- RadioButton
- Static
- Text box
- UserControl

The purpose of the UserControl class is to provide the user the ability to define customized controls that cannot be defined using the standard control classes. For example, you can use this class to define ActiveX objects, or unique HTML controls. A UserControl must have a name and must be an XML element.

Container Controls

- GroupBox
- Window

Table Controls

- Table
- TableHeader

Context Package

The context package contains the **EventContext** class.

This is an abstract class which contains the following methods:

- **`getSubApplName, getLibraryName, getApplicationName`** - retrieve the name of the current Subapplication, library or Application.
- **`getBrowserType`** - retrieves the current browser type.
- **`getSharedUserVariable, writeSharedUserVariable, deleteSharedUserVariable`** - used to work with shared user variables within Java extensions. Shared user variables are used in ACE methods as a means of transferring information between Subapplications in an Application.
- **`getUserPerSessionData`** - used to access a hashtable object named **`userPerSessionData`**.

  The XHTML Processing Module provides access to a hashtable object, named userPerSessionData, which you can use to store and retrieve data for a specific session. You use standard Hashtable methods to store and retrieve the data. This is useful, for example, for saving data entered in a specific screen so that upon the next entry to this same screen the information already appears in the fields.

Example:

When submitting the LOGON page, you may want to save the User ID and Password details for the current session. Thus, upon the next entry to the LOGON screen, these details will automatically appear in their corresponding fields.

To implement this, you need write a Subapplication scope extension in which the User ID and Password are saved in the onPageSubmit code, and retrieved in the onPageLoad code.

The **`EventContext`** class is the parent of the following event-specific classes:

- `OnPageLoadContext`
- `OnPageSubmitContext`
- `OnControlReadyContext` - for more information, see “Java API - onControlReady” on page 292.

onPageLoadContext

The **`onPageLoadContext`** class extends the **`EventContext`** class. In addition to the methods it inherits from **`EventContext`**, **`onPageLoadContext`** contains the following methods:

- **`addCookie`**, **`getCookie`** - add or retrieve a specific cookie object.
- **`getWindow`** - retrieves the current window control.

onPageSubmitContext

The **`onPageSubmitContext`** class extends the **`EventContext`** class. In addition to the methods it inherits from **`EventContext`**, **`onPageSubmitContext`** contains the following methods:

- **`addCookie`**, **`getCookie`** - add or retrieve a specific cookie object.
- **`getPostData`** - retrieves the post-data hashtable, which contains the user data that is sent from the client. Use this method when you need to manipulate the user data before it is processed by the server.

Control-level Extensions - onControlReady

The **`onControlReady`** event is used to perform control-level extensions. This event enables you to write extensions that affect the properties of a particular control type across an entire Application. For example, you can replace all the date controls in the Application with a customized date control.

This event occurs after the updating of the static XHTML with the dynamic runtime information. For each control, after it is ready, a Java extension is searched for. If a Java extension exists for this control type, it is applied. In this manner, the extension is applied to all the controls of a certain type.

Note: The `onControlReady` event is activated before the `onPageLoad` event. This means that the `onPageLoad` event may still change properties set by `onControlReady`.

The following extension capabilities are available:

- Setting (set) or retrieving (get) control properties.

  For example, you can set/get control properties such as: size, location, text, fonts, colors, show/hide, enable/disable, balloon help, Z-ordering, etc.
- Replacing existing controls with new controls.
- Adding JavaScript event handlers for each control.

  For example: `onClick`, `onMouseOver`, `onFocus`, `onKeyDown`, `onLoad`, etc.
- Adding JavaScript functions to be activated by one of the event handlers above.

  For example, a JavaScript function to be activated by the `onClick` event attached to a button.

Applying Skins Across Applications

Control-level extensions are used to apply different skins to your Applications. The term “skin” is used to reference a specific look and feel that you apply to the user interface of your Application. Skins may affect graphic elements such as the background of your Application, control color schemes, graphics, etc.

In contrast to page-level extensions, which are Application specific, control-level extension files are Application independent. You can apply control-level extensions across different Applications.

Setting Up an Application to Use Skins

Setting up an Application to use skins involves two main steps:

- Writing the control-level extension files
- Configuring the Application INI file

Note: You can also set and remove skins via DoMethods in ACE. For more information, see “Skin-related DoMethods” on page 292.

Writing Control-level Extension Files

Control-level extensions must be written in files with specific names, according to the control type.

The following file names must be used:

- `ButtonExtension.java`
- `CheckBoxExtension.java`
- `ComboBoxExtension.java`
- `DateExtension.java`
- `GroupBoxExtension.java`
- `LineExtension.java`
- `LinkExtension.java`
- `MultilineTextBoxExtension.java`
- `PromptExtension.java`
- `RadioButtonExtension.java`
- `StaticExtension.java`
- `TableExtension.java`
- `TableHeaderExtension.java`
- `TextBoxExtension.java`
- `WindowExtension.java`

The location of the control-level extension files can be determined by the user, as long as it is in the server classpath.

To modify the classpath, use the following guidelines:

- Development environment - anywhere under the following directory: `<InstallDir>\JacadaFiles\`
- Deployment environment - anywhere under the runtime installation directory.

Configuring the Application INI File

To use control-level extensions, you need to configure the Application INI file.

The Application INI file is named according to the Application, and is located in the following directory of the development environment:
`<InstallDir>\appls\<ApplName>\rt32\<ApplName>.INI`

To configure the Application INI file to use control-level extensions, or skins, follow these steps:

- Add a new section - `[Skins]`
- Within the `[Skins]` section, define all the skins to be used in this Application, as follows:

  ```
  skinName1=<packagename for skin1>
  skinName2=<packagename for skin2>
  ```

  For example:

  ```
  skinName1=skins.red
  skinName2=skins.green
  ```

  The package specified must correspond to the location of the extension files.
- Specify the active skin, as follows:

  ```
  Active = skinName1
  ```

  In order for the extensions in the package to be activated when the Application is loaded, the active skin name value must be a valid name from the list. Otherwise, none of the extensions are activated.

Example 14. Use of skins - <ApplName>.ini settings

The runtime installation of Application A1 is at `c:\JIS`. The directory `c:\JIS\skins\` contains two subdirectories, red and green, in which extension files were placed. These extension files contain the properties of each one of the skins. The current active skin is green. Below is an example of the `[Skins]` section in the `A1.ini` file.

```
[Skins]
red = skins.red
green = skins.green
Active = green
```

Skin-related DoMethods

In order to enable referencing skins within ACE methods, the following DoMethods are provided:

- ***`SetSkin`*** - sets the current active skin to be used in the Application.

  Executed by the Application

  Uses one parameter - skinName, of type string. The value of this parameter must correspond to one of the skin names in the Application INI file

  Returns _TRUE if successful, _FALSE otherwise.
- ***`RemoveSkin`*** - removes the current active skin, leaving the default look and feel that was set in ACE.

  Executed by the Application.

  Returns _TRUE if successful, _FALSE otherwise.

  If no skin was set, returns _TRUE.

Java API - onControlReady

A Java API is provided to enable you to write extensions. The full details of the Java API are described in a Javadoc that is provided with the product. This section, however, reviews a few highlights of the following package included in the API:

`com.jacada.jis.runtime.server.frontend.xhtml.context`

The context package contains the **`EventContext`** class, which contains a set of methods. For details on these methods, see “Context Package” on page 288.

The **`EventContext`** class is the parent of the **`OnControlReadyContext`** class. The **`OnControlReadyContext`** class contains the following methods:

- **`getControlType`** - retrieves the type of the current control.
- **`getXHTMLControl`** - retrieves the XHTML control.

Example 15. Extending the static controls

This example demonstrates an extension file for extending the Static controls in an Application to have green text. The package which contains the extension files is `skins.green`. The file name is `StaticExtension.Java`.

```
package skins.green;
import com.jacada.jis.runtime.server.frontend.xhtml.extension.
IControlExtension;
import com.jacada.jis.runtime.server.frontend.xhtml.context.
OnControlReadyContext;
import com.jacada.jis.runtime.server.frontend.xhtml.controls.*;
public class StaticExtension implements IControlExtension
{
   public void onControlReady(OnControlReadyContext context) {
   Static  myStaticControl = (Static)context.getXhtmlControl();
   myStaticControl.setForeground("green");
   }
}
```

Java Extension Examples

To illustrate the advantages gained through attaching Java extensions, the following code examples are presented and explained in this section:

- COOKIES
- BROWSERS
- BADINPUT
- RETRIEVE
- SKINDEMO

Note: Each one of the names of these examples is the name of a JIS Application created for the purpose of illustrating the example. Hence the upper-case.

COOKIES

This example shows:

- How to write a Subapplication extension.
- How to **create** a cookie.
- How to **set** and **retrieve** a cookie's value.
- How to find a specific control in a page and set this control's text.
- How to read PostData.

In this example, the last user name is saved and then loaded the next time the browser loads this page. A Subapplication-level extension named `"LOGON.java"` is associated with the `LOGON` Subapplication.

LOGON.java

This Java extension is associated with the `LOGON` Subapplication. The instructions in this file are executed every time the `LOGON` Subapplication is entered.

The LOGON class in this file contains two methods:

- `onPageLoad`
- `onPageSubmit`

onPageLoad

This method:

- Checks if the `"CookiesApplicationUserName"` cookie exists.
- If the cookie exists, the method retrieves the cookie and assigns its value into the `"user_text_box"` text box.

onPageSubmit

This method:

- Retrieves the value of the `"user_text_box"` text box.
- Creates a cookie with that value.
- Saves the cookie.

Note: These actions are pointed out in the following code.

Following is the code for the `LOGON.java` file:

```
package appls.COOKIES.xhtml.user;

import javax.servlet.http.Cookie;
import com.jacada.jis.runtime.server.frontend.xhtml.context.*;
import com.jacada.jis.runtime.server.frontend.xhtml.controls.*;
import java.util.Hashtable;
public class LOGON extends Appl {
   public void onPageLoad(OnPageLoadContext context) {
      super.onPageLoad(context);
      Cookie cookie = context.getCookie
         ("CookiesApplicationUserName");
      if (cookie != null) { ]
         String value = cookie.getValue();
         if (value != null) {
            Window window = context.getWindow();
            XhtmlControl textBox = window.getControlByName
            ("user_text_box");
            textBox.setText(value);
         }
      }
   }

   public void onPageSubmit(OnPageSubmitContext context) {
      super.onPageSubmit(context);
      Hashtable postData = context.getPostData();
      String name = "txt_user_text_box";
      String[] value = (String[]) postData.get(name);
      if (!(value[0].equals(""))) {
         Cookie cookie = new Cookie
         ("CookiesApplicationUserName", value[0]);
         cookie.setMaxAge(Integer.MAX_VALUE);
         context.addCookie(cookie);
      }
   }
}
```

Imports

Retrieve Cookies

Create Cookie

Save Cookie

LOGON

onPageLoad

onPageSubmit

BROWSERS

This example shows:

- How to write a Subapplication extension.
- How to check browser type.
- How to create a user control and add it to a page.
- How to set a background image for a page.

In this example, varying XHTML elements are added into a specific page, depending on the browser type. To accomplish this, a Subapplication-level Java extension is associated with the Application's `LOGON` Subapplication.

LOGON.java

This Java extension is associated with the `LOGON` Subapplication.

The **`onPageLoad`** method creates the appropriate XHTML element for setting the window's background music, and adds this element to the page. The XHTML element used for this is dependent on the browser type in that:

- If the browser type is **Netscape**, the following XHTML element is created:
  `<EMBED SRC="sound/strauss.mid" width="0" height="0" loop="true" autostart="true" />`
- If the browser type is **Microsoft Internet Explorer**, the following XHTML element is created:
  `<BGSOUND SRC="sound/strauss.mid" LOOP="infinite"/>`

The **`onPageLoad`** method also sets a background image for the page irrespective of the browser type.

Following is the code for the `LOGON.java` file:

```
package appls.BROWSERS.xhtml.user;
```

Imports

```
import org.w3c.dom.*;
import javax.xml.parsers.*;
import com.jacada.jis.runtime.server.frontend.xhtml.extension.*;
import com.jacada.jis.runtime.server.frontend.xhtml.context.*;
import com.jacada.jis.runtime.server.frontend.xhtml.controls.*;
import com.jacada.jis.runtime.server.frontend.xhtml.controls.general.*;
import com.jacada.jis.runtime.server.frontend.xhtml.general.*;
public class LOGON extends Appl {

   public void onPageLoad (OnPageLoadContext context) {
      Window window = context.getWindow();
```

LOGON

onPageLoad

Set back-ground image

Get browser type

Browser MSIE

Browser Netscape

Insert new element

```
        window.setImage("/images/emusic.gif");
            String browserType = context.getBrowserType();
            Element el = null;
        DocumentBuilderFactory dFactory =
        DocumentBuilderFactory.newInstance();
        DocumentBuilder dBuilder = null;
        Document doc = null;
        try {
            dBuilder  = dFactory.newDocumentBuilder();
            doc = dBuilder.newDocument();
        }
        catch (Exception e) {
            e.printStackTrace();
            return;
        }
        if  (browserType.equals(BrowserTypes.MSIE) ||
        browserType.equals(BrowserTypes.MSIE4)) {
            el = doc.createElement("BGSOUND");
            el.setAttribute("SRC", "sound/strauss.mid");
            el.setAttribute("LOOP", "infinite");
        }
        else if (browserType.equals(BrowserTypes.Netscape)) {
            el = doc.createElement("EMBED");
            el.setAttribute("SRC", "sound/strauss.mid");
            el.setAttribute("width", "0");
            el.setAttribute("height", "0");
            el.setAttribute("loop", "true");
            el.setAttribute("autostart", "true");
        }
        if (el != null) {
            UserControl newControl = new UserControl("myMusic", el);
            window.appendChild(newControl);
        }
    }
}
```

LOGON

onPageLoad

BADINPUT

This example shows:

- How to read and change PostData.
- How to add a JavaScript to a page.
- How to set an event handler to a control.

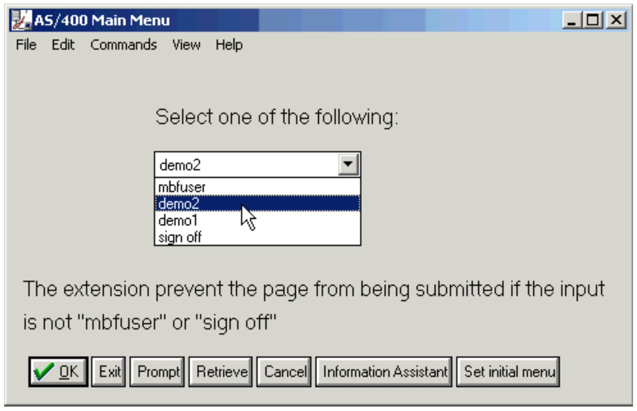
How to avoid posting data.

To accomplish this, two Subapplication-level Java extensions are added:

- MAINMENU.java
- MBF001.java

MAINMENU.java

This Java extension is associated with the `MAINMENU` Subapplication:



Select one of the following:

The extension prevent the page from being submitted if the input is not "mbfuser" or "sign off"

Figure 64. The MAINMENU subapplication

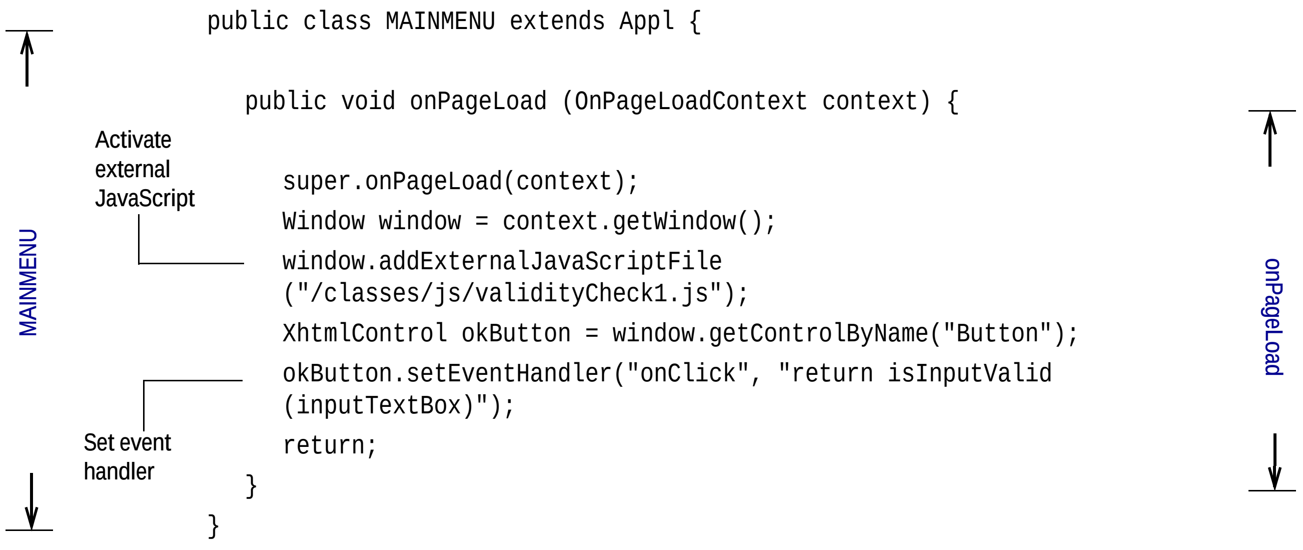
The Java extension contains a reference to the `validityCheck1.js` JavaScript file. The JavaScript contains the **`isInputValid()`** method, which performs a validity check on the option the end-user selects from the combo box. The **`onPageLoad`** method connects the **OK** button's `"onClick"` event to the JavaScript file's `"isInputValid(inputTextBox)"` method. If the method returns false, then the page is not submitted.

Following is the code for the `MAINMENU.java` file:

```
package appls.BADINPUT.xhtml.user;
```

Imports

```
import java.util.*;
import com.jacada.jis.runtime.server.frontend.xhtml.extension.*;
import com.jacada.jis.runtime.server.frontend.xhtml.context.*;
import com.jacada.jis.runtime.server.frontend.xhtml.controls.*;
import com.jacada.jis.runtime.server.frontend.xhtml.general.*;
```



```
public class MAINMENU extends Appl {

   public void onPageLoad (OnPageLoadContext context) {

      super.onPageLoad(context);
      Window window = context.getWindow();
      window.addExternalJavaScriptFile
      ("/classes/js/validityCheck1.js");
      XhtmlControl okButton = window.getControlByName("Button");
      okButton.setEventHandler("onClick", "return isInputValid
      (inputTextBox)");
      return;
   }
}
```

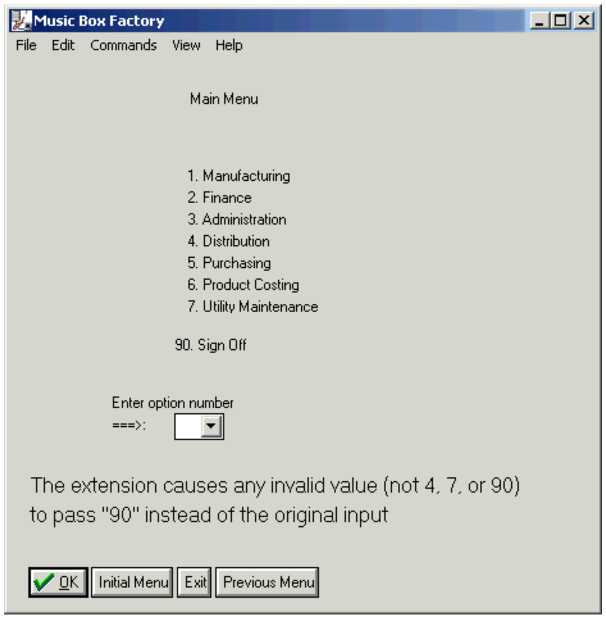
validityCheck1.js

Following is the code for the `validityCheck1.js` file. This is the JavaScript file referred to in the `MAINMENU.java` file for performing the validity check:

```
function isInputValid(elem) {
   var index =  elem.selectedIndex;
   if (index == -1) {
      return false;
   }
   input = elem.options[index].text;
   if (input == 'mbfuser' || input == 'sign off') {
      return true;
   }
   return false;
}
```

MBF001.java

This Java extension is associated with the `MBF001` Subapplication:



The extension causes any invalid value (not 4, 7, or 90) to pass "90" instead of the original input

Figure 65. The MBF001 subapplication

In this Java extension, the `onPageSubmit` method looks in the `postData` to see if the input value of the combo box is a valid value. A valid value can only be one of the following:

- `"4"`
- `"04"`
- `"7"`
- `"07"`
- `"90"`

If the value is an invalid value, then this value is replaced with `"90"`.

Following is the code for the `MBF001.java` file:

```
package appls.BADINPUT.xhtml.user;
```

Imports

```
import java.util.*;
import com.jacada.jis.runtime.server.frontend.xhtml.extension.*;
import com.jacada.jis.runtime.server.frontend.xhtml.context.*;

public class MBF001 extends Appl {

   static Set validOptions;
```

Valid options

MBF001

onPageSubmit

Replace invalid data in PostData

```
    static {
       validOptions = new HashSet();
       validOptions.add("4");
       validOptions.add("04");
       validOptions.add("7");
       validOptions.add("07");
       validOptions.add("90");
    }
    public void onPageSubmit (OnPageSubmitContext onSubmitContext) {
       super.onPageSubmit(onSubmitContext);
       Hashtable postData = onSubmitContext.getPostData();
       Enumeration paramNames = postData.keys();
       String name;
       while (paramNames.hasMoreElements()) {
          name = (String) paramNames.nextElement();
          if (name.equals("cbo_DDSOPTION")) {
             String []  value = (String[]) postData.get(name);
             String optionString = value[0];
             if (validOptions.contains(optionString)) {
             }
             else {
                value[0] = "90";
             }
             break;
          }
       }
       return;
    }
}
```

RETRIEVE

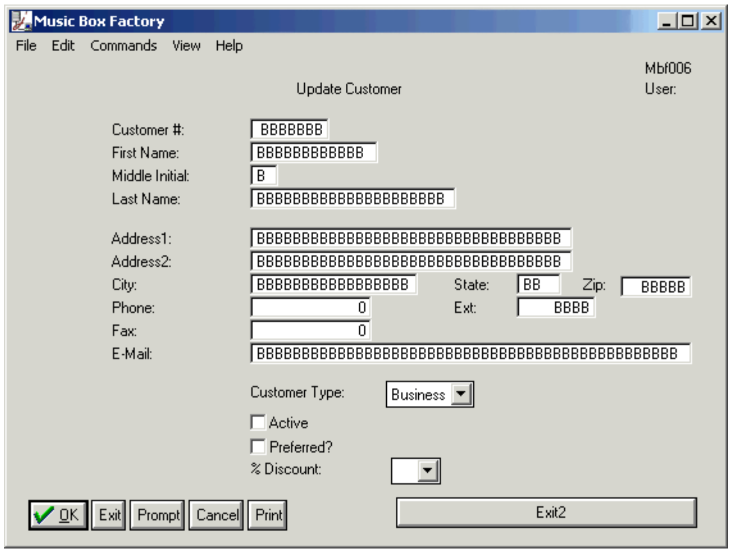
This example shows:

- How to write a Subapplication-level Java extension.
- How to use PostData.
- How to use "User Per Session Data".

This example demonstrates how you can save the information typed on a page and retrieve this information later when re-entering that specific page. This might prove useful, for example, when a spontaneous screen appears on the host which causes the data entered by the end-user to be erased. Using a Subapplication-level Java extension, you can prevent this data from being lost.

MBF006R1.java

This Java extension is associated with the MBF006R1 Subapplication:



Mbf006
User:

Update Customer

Customer #: BBBBBBB
First Name: BBBBBBBBBBBB
Middle Initial: B
Last Name: BBBBBBBBBBBBBBBBBBBBB

Address1: BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
Address2: BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
City: BBBBBBBBBBBBBBBBB State: BB Zip: BBBBB
Phone: 0 Ext: BBBB
Fax: 0
E-Mail: BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB

Customer Type: Business
☐ Active
☐ Preferred?
% Discount:

Figure 66. The MBF006R1 subapplication

This Java extension uses the following methods:

- `onPageLoad`
- `onPageSubmit`

onPageLoad

This method prepares a key from the data in the Customer ID text box and does the following.

- If `"userPerSessionData"` previously contains an entry with this key:
  - The current key entry is removed.
  - The key's data is retrieved.
  - The controls of the Subapplication are filled with their corresponding data.
- Writes the `"myKeyIs"` entry to the `"userPerSessionData"`, so when the `"onSubmit"` occurs, it will know which key to use.

onPageSubmit

This method creates a new `Hashtable` object called `"dataToKeep"` and goes over all post data in a loop which does the following:

- Keeps the data from text boxes, combo boxes, and check boxes in the `"dataToKeep"` object.
- If the end-user has pushed the **Exit** or the **Cancel** buttons, then the loop is broken.
- At the end of the loop, the `"myKeyIs"` entry is read from the `"userPerSessionData"` in order to know which key to use. This key is used for keeping the `"dataToKeep"` in the `"userPerSessionData"`.

Following is the code for the `MBF006R1.java` file:

```
package appls.RETRIEVE.xhtml.user;
```

Imports

```
import java.util.*;
import com.jacada.jis.runtime.server.frontend.xhtml.context.*;
import com.jacada.jis.runtime.server.frontend.xhtml.controls.*;
import com.jacada.jis.runtime.server.frontend.xhtml.general.*;
public class MBF006R1 extends Appl {

   public void onPageLoad (OnPageLoadContext context) {

      super.onPageLoad(context);
      Window window = context.getWindow();
      Hashtable upsd = context.getUserPerSessionData();
      XhtmlControl customerIdControl = window.getControlByName
      ("DDSDCUSTN");
      String key = customerIdControl.getText();
      upsd.put("MyKeyIs", key);
      Hashtable retrievedData = (Hashtable) upsd.remove(key);
      if (retrievedData == null) {
         return;
      }
      Enumeration paramNames = retrievedData.keys();
      String controlName;
      String controlValue;
      XhtmlControl inputControl;

      while (paramNames.hasMoreElements()) {
         controlName = (String) paramNames.nextElement();
         controlValue = (String) retrievedData.get(controlName);
         inputControl = window.getControlByName(controlName);
```

MBF006R1

onPageLoad

```
      if (inputControl != null) {
         if (inputControl.getJacadaControlType().equals
         (JacadaControlTypes.CHECKBOX_CTRL)) {
            boolean checked = (controlValue.equals
            (String.valueOf(true)));
            ((CheckBox) inputControl).setChecked(checked);
         }
         else {
            inputControl.setText(controlValue);
         }
      }
   }
   return;
}
public void onPageSubmit (OnPageSubmitContext context) {

   super.onPageSubmit(context);
   Hashtable postData = context.getPostData();
   Hashtable dataToKeep = new Hashtable();
   String name;
   String[] value;
   String textPrefix = "txt_";
   String comboPrefix = "cbo_";
   String checkboxPrefix = "chk_";
   String buttonPrefix = "btn_";
   Enumeration paramNames = postData.keys();
   while (paramNames.hasMoreElements()) {
      name = (String) paramNames.nextElement();
      System.out.println(name);
      value = (String[]) postData.get(name);
      if (name.startsWith(textPrefix)) {
         dataToKeep.put(name.substring(textPrefix.length()),
         value[0]);
      }
      else if (name.startsWith(comboPrefix)) {
         dataToKeep.put(name.substring(comboPrefix.length()),
         value[0]);
      }
      else if (name.startsWith(checkboxPrefix)) {
         boolean checked = (value.length == 2);
         dataToKeep.put(name.substring(checkboxPrefix.length()),
         String.valueOf(checked));
```

onPageLoad

onPageSubmit

MBF006R1

MBF006R1

onPageSubmit

If EXIT button is pressed

If CANCEL button is pressed

```
         }
         else if (name.startsWith(buttonPrefix)) {
            if (name.substring(buttonPrefix.length()).equals
            ("Exit")) {
               return;
            }
            if (name.substring(buttonPrefix.length()).equals
            ("Sub2202301006")) {
               return;    }
         }
      }
      Hashtable upsd = context.getUserPerSessionData();
      String key = (String) upsd.get("MyKeyIs");
      upsd.put(key, dataToKeep);
      return;
   }
}
```

SKINDEMO

This example shows:

- How to set the Runtime INI file to enable customized skins.
- Some skin code examples.
- How to change control attributes.

Runtime INI Settings

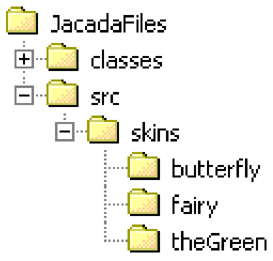
The `[skins]` section in the `SKINDEMO.ini` file is as follows:

```
[skins]
skin1=skins.butterfly
skin2=skins.fairy
skin3=skins.theGreen
;Active=skin1
```

In this example:

- "skin1", "skin2" and "skin3" are skin names.
- `"skin.butterfly"`, `"skin.fairy"` and `"skin.theGreen"` are skin package names.

  The skin packages are placed in their respective directories as seen below:



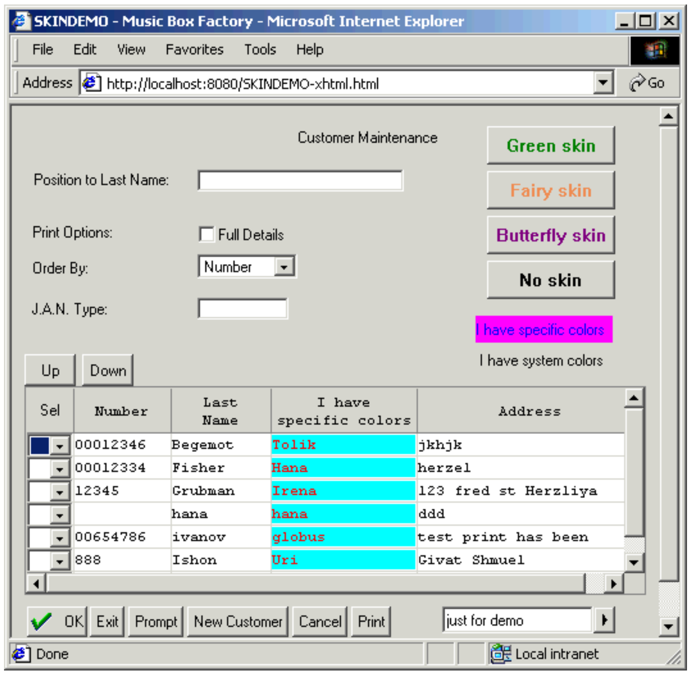
```
JacadaFiles
  classes
  src
    skins
      butterfly
      fairy
      theGreen
```

Figure 67. Skins directory structure

- The active skin is "skin1", however this code line is commented out.

The "skin.butterfly" Skin Package

Following is one of the SKINDEMO application’s screens:



SKINDEMO - Music Box Factory - Microsoft Internet Explorer

File Edit View Favorites Tools Help

Address http://localhost:8080/SKINDEMO-xhtml.html Go

Customer Maintenance

Green skin | Fairy skin | Butterfly skin | No skin

Position to Last Name:

Print Options: Full Details

Order By: Number

J.A.N. Type:

I have specific colors

I have system colors

Up Down

Sel	Number	Last Name	I have specific colors	Address
	00012346	Begemot	Tolik	jkhjk
	00012334	Fisher	Hana	herzel
	12345	Grubman	Irena	123 fred st Herzliya
		hana	hana	ddd
	00654786	ivanov	globus	test print has been
	888	Ishon	Uri	Givat Shmuel

OK Exit Prompt New Customer Cancel Print just for demo

Done Local intranet

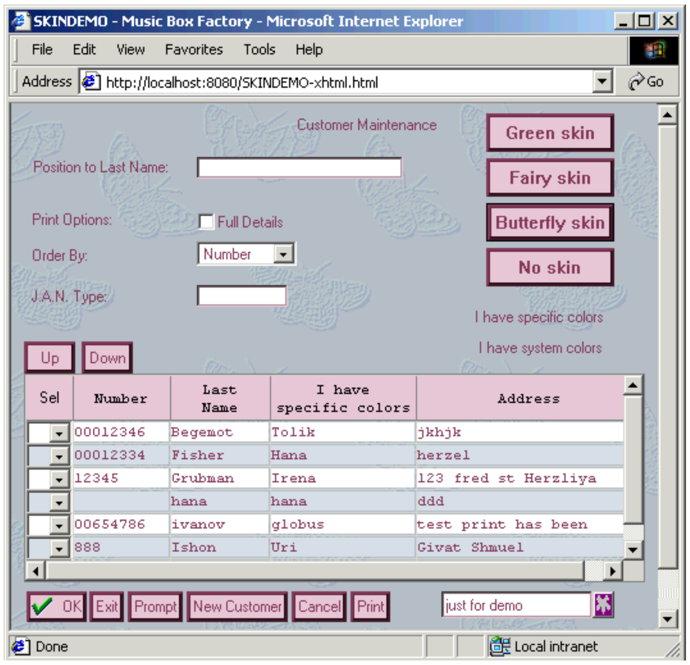
Figure 68. The SKINDEMO screen “before”

The screen includes the following control types:

- Button
- Prompt
- Table
- TableHeader
- Combo box
- Check box

- Text box
- Static

Applying the "`skin.butterfly`" skin package results as follows:



http://localhost:8080/SKINDEMO-xhtml.html

Customer Maintenance

Position to Last Name:

Print Options: Full Details

Order By: Number

J.A.N. Type:

I have specific colors

I have system colors

Sel	Number	Last Name	I have specific colors	Address
	00012346	Begemot	Tolik	jkhjk
	00012334	Fisher	Hana	herzel
	12345	Grubman	Irena	123 fred st Herzliya
		hana	hana	ddd
	00654786	ivanov	globus	test print has been
	888	Ishon	Uri	Givat Shmuel

just for demo

Figure 69. The SKINDEMO screen "after"

In this screen each control type is differently treated in that:

- Window control has a butterfly wallpaper decorating its background.
- Static text is painted purple with a transparent background.
- Button controls are painted pink and have a thicker border.
- Prompt controls have an image of a flower on their button.
- Table headers are painted pink and their text is black.
- Table rows are alternately painted blue and white.

The "`skin.butterfly`" skin package includes the following Java files:

- Colors.java
- XhtmlControlExtension.java
- StaticExtension.java
- ButtonExtension.java
- PromptExtension.java
- TableExtension.java
- TableHeaderExtension.java
- EditExtension.java

- WindowExtension.java
- PromptExtension.java

Colors.java

This Java extension is not specific to a control type. The Colors class in this Java extension contains a color conversion table that defines color names. The color names are used in control extensions for color definition settings.

Following is the code for the `Colors.java` file:

Colors

Color conversion table

```
package skins.butterfly;
import java.util.Hashtable;

class Colors {

   public static String lightPink;
   public static String darkPink;
   public static String white;
   public static String black;
   public static String automatic;
   public static String transparent;
   public static String lightBlue;

   static {
      lightBlue = "#D2DCE6";
      lightPink = "#E3C4D2";
      darkPink =  "#853F5F";
      white =  "white";
      black =  "black";
      automatic = "Automatic";
      transparent = "transparent";
   }
}
```

XhtmlControlExtension.java

This Java extension is not associated with a specific control. However, other control extensions, such as **`StaticExtension`**, extend this Java extension's **`XhtmlControlExtension`** class. In a control extension that extend this class, all settings from this class apply, except for settings in the control extension, which override the settings in the **`XhtmlControlExtension`** class. The **`onControlReady`** method in this class sets color properties.

Following is the code for the `XhtmlControlExtension.java` file:

Imports

Set color properties

XhtmlControlExtension

onControlReady

```
package skins.butterfly;

import com.jacada.jis.runtime.server.frontend.xhtml.extension.
IControlExtension;
import com.jacada.jis.runtime.server.frontend.xhtml.context.
OnControlReadyContext;
import com.jacada.jis.runtime.server.frontend.xhtml.controls.
XhtmlControl;

public class XhtmlControlExtension implements com.jacada.jis.
runtime.server.frontend.xhtml.extension.IControlExtension {

   public void onControlReady(OnControlReadyContext context) {

      XhtmlControl control = context.getXhtmlControl();
      control.setBackground(backgroundColor());
      control.setForeground(foregroundColor());
      control.setBorderColor(borderColor());
      return;
   }
   String backgroundColor() {
      return Colors.transparent;
   }
   String foregroundColor() {
      return Colors.darkPink;
   }
   String borderColor() {
      return Colors.automatic;
   }
}
```

StaticExtension.java

This Java extension is associated with Static controls. The only function of the **`StaticExtension`** class is to extend the **`XhtmlControlExtension`** class, as mentioned earlier.

Following is the code for the `StaticExtension.java` file:

```
package skins.butterfly;

public class StaticExtension extends XhtmlControlExtension {
}
```

ButtonExtension.java

This Java extension is associated specifically with Button controls. The **`onControlReady`** method sets Button controls' border width and color definitions.

Following is the code for the `ButtonExtension.java` file:

```
package skins.butterfly;
```

Imports

```
import com.jacada.jis.runtime.server.frontend.xhtml.context.
OnControlReadyContext;
import com.jacada.jis.runtime.server.frontend.xhtml.controls.
XhtmlControl;
```

ButtonExtension

onControlReady

Set border width

Set color properties

```
public class ButtonExtension extends XhtmlControlExtension {

  public void onControlReady(OnControlReadyContext context) {

    super.onControlReady(context);
    XhtmlControl control = context.getXhtmlControl();
    control.setBorderWidth(3);
    return;
  }
  String backgroundColor() {
    return Colors.lightPink;
  }
  String borderColor() {
    return Colors.darkPink;
  }
}
```

TableExtension.java

This Java extension is associated with Table controls. **`onControlReady`** sets color properties in such a way that table rows are alternately colored blue and white.

Following is the code for the `TableExtension.java` file:

```
package skins.butterfly;
```

Imports

```
import com.jacada.jis.runtime.server.frontend.xhtml.context.
OnControlReadyContext;
import com.jacada.jis.runtime.server.frontend.xhtml.controls.Table;
import com.jacada.jis.runtime.server.frontend.xhtml.controls.
XhtmlControl;

public class TableExtension extends XhtmlControlExtension {

   public void onControlReady(OnControlReadyContext context) {

      super.onControlReady(context);
      Table table = (Table) context.getXhtmlControl();
      String colors [] = {Colors.white, Colors.lightBlue};
      XhtmlControl currControl;
      String newBackgroundColor;
      String newForegroundColor = Colors.darkPink;

      for (int i = 0, numOfRows = table.getRowsNumber();
      i < numOfRows; i++) {

         newBackgroundColor = colors[i % 2];
         for (int j = 0, numOfColumns = table.getColumnsNumber();
         j < numOfColumns; j++) {

            currControl = table.getCell(i, j);
            currControl.setBackground(newBackgroundColor);
            currControl.setForeground(newForegroundColor);
         }
      }
      return;
   }
}
```

Set Table color properties

TableExtension

onControlReady

TableHeaderExtension.java

This Java extension is associated with TableHeader controls. The **`TableHeaderExtension`** class sets background color and border color.

Following is the code for the `TableHeaderExtension.java` file:

```
package skins.butterfly;

public class TableHeaderExtension extends XhtmlControlExtension {

   String backgroundColor() {
      return Colors.lightPink;
   }
```

Set background color

TableHeaderExtension

```
    String borderColor() {
        return Colors.darkPink;                    Set border
    }                                              color
}
```

EditExtension.java

This Java extension is associated with Edit box controls. The **`EditExtension`** class contains background and border color settings.

Among the control extensions that extend this class are:

- `PromptExtension`
- `WindowExtension`

Following is the code for the `EditExtension.java` file:

EditExtension

```
package skins.butterfly;

public abstract class EditExtension extends XhtmlControlExtension {

    String backgroundColor() {
        return Colors.white;
    }
    String borderColor() {
        return Colors.lightPink;
    }
}
```

WindowExtension.java

This Java extension is associated with Window controls. The **`WindowExtension`** class in this Java extension extends **`EditExtension`**, as mentioned earlier, in "EditExtension.java" on page 312.

The **`onControlReady`** method sets the `butterfly.gif` file as the window's background image.

Following is the code for the `WindowExtension.java` file:

Imports

```
package skins.butterfly;

import java.util.Vector;
import com.jacada.jis.runtime.server.frontend.xhtml.context.
OnControlReadyContext;
import com.jacada.jis.runtime.server.frontend.xhtml.controls.*;
```

WindowExtension

onControlReady

Set back-ground image

```
public class WindowExtension extends EditExtension {

   public void onControlReady(OnControlReadyContext context) {

      super.onControlReady(context);

      Window window = (Window) context.getXhtmlControl();

      window.setImage("/images/butterfly.gif");

      return;
   }
}
```

PromptExtension.java

This Java extension is associated with Prompt controls. The **`PromptExtension`** class in this Java extension extends from **`EditExtension`**, as mentioned earlier in "EditExtension.java" on page 312

The **`onControlReady`** method sets an image for the control's button.

Following is the code for the `PromptExtension.java` file:

```
package skins.butterfly;
```

Imports

```
import com.jacada.jis.runtime.server.frontend.xhtml.context.
OnControlReadyContext;
import com.jacada.jis.runtime.server.frontend.xhtml.controls.*;
```

PromptExtension

onControlReady

Set button image

```
public class PromptExtension extends EditExtension {

   public void onControlReady(OnControlReadyContext context) {

      super.onControlReady(context);

      Prompt prompt = (Prompt) context.getXhtmlControl();

      prompt.setImage("/images/prompt_flower.gif");

      return;
   }
}
```

Extensions to the Date Calendar Window

The purpose of this section is to describe the use of the `DateExtension` java extension to modify the appearance of the Date Calendar Window. The procedures discussed in this section apply specifically to extending the Date Calendar window, but can be used as a model for extending any of the other controls listed on page 287.

When you have completed this section, you should be able to use the `DateExtension` class to change the appearance of the Date Calendar Window.

The Date Calendar Window

Attached to the default Date control in ACE is a Calendar Window. It is activated by pressing the calendar icon next to the Date control, as shown in Figure 70.

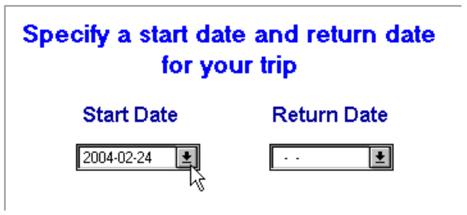


Figure 70. The date control

The Calendar Window itself is shown in Figure 71.

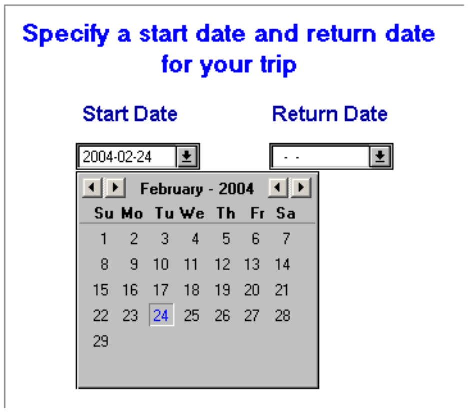


Figure 71. The date calendar window

Changing the Appearance of the Date Calendar Window

Figure 71 shows the default appearance of the Date Calendar Window. You can change its appearance through the use of the `DateExtension.java` class.

Types of Changes Available

The changes that can be made to the Calendar Window's appearance and the associated method syntax are listed below.

- The week can be changed to start with a day other than Sunday. Use an integer to indicate the desired day. 2=Monday, 3=Tuesday, and so on. For example, to change the calendar to show Monday as the first day of the week, code: `date.setCalendarFirstWeekDay(2);`
- The background color can be modified. Standard java colors can be specified. For example, to change the background color of the Calendar Window to orange, code: `date.setCalendarBackgroundColor(“orange”);`
- The color of the calendar text and dates can be modified. Specify a standard java color. For example, to change the color of the calendar text and dates to green, code: `date.setCalendarColor(“green”);`
- The text font used in the calendar can be modified. Use one the java standard font families. For example: `date.setCalendarFontFamily(“serif”);`
- The color of the selected day in the calendar can be changed. Use a standard java color. For example, to change the color of the selected day to blue, code: `date.setCalendarSelectedDayColor(“blue”);`
- The text in the Calendar Window's title bar can be changed. For example: `date.setCalendarTitle(“mytitle”);`
- You can specify the initial location of the window, in pixels. For example: `date.setCalendarWindowLocation(10,10);`
- The line object that sits over and under the days of the week can be changed. Point to the object you wish to use:
  ```
  date.setCalendarLineGif(“http://localhost:8081/images/myline.gif”)
  ```
- The names of the months can be changed. Separate the names with an asterisk. `date.setCalendarMonths(“jan*fev*mar*avr*mai*jui*jui*aou* sep*oct*nov*dec”)`
- You can change the button images used for navigating to Next Month, Next Year, Previous Month, Previous Year.
  ```
  date.setCalendarNextMonthGif(“http://localhost:8081/images/mybutton1.gif”)
  date.setCalendarNextYearGif(“http://localhost:8081/images/mybutton1.gif”)
  date.setCalendarPrevMonthGif(“http://localhost:8081/images/mybutton1.gif”)
  date.setCalendarPrevYearGif(“http://localhost:8081/images/mybutton1.gif”)
  ```

Scope of Changes

The directory location of the `DateExtension` class determines its scope. The `DateExtension.java` class can be applied across an entire application, for a particular library, or for a specific subapplication. This class can also be used as part of an application skin. The simplest approach is to use an application skin (See "Modifying the Date Calendar Window in a Skin" on page 320), but for the sake of completeness we also provide examples of how to apply the `DateExtension` class application-wide, library-wide, and to a specific Subapplication.

The package name and the location of the extension determine the scope of the extension, as explained in continuation.

If you have extensions of different scopes defined for a given application, only one of the extensions will be applied. The priority of extensions, from strongest to weakest, is: Subapplication scope, Library scope, Application scope.

Application Scope

If you want your extension code to affect an entire application, the extension file must be named `Appl.java`; the package name must be `appls.<ApplName>.xhtml.user`

The runtime generation process generates the following two files automatically:

`Appl.java` - a skeleton file in which you insert your code.

`jacc.bat` - a batch file used to compile the code in the appl.java file.

The above two files are generated in the directory:
`<InstallDir>\jacadafiles\src\appls\<ApplName>\xhtml\user`

Use `jacc.bat` to compile the Java source code. The resulting classes are placed in the application classes directory:
`<InstallDir>\jacadafiles\classes\appls\<ApplName>\xhtml\user`

For an application named APPL1, this is the code in the `Appl.java` file:

```
//--------------------------------------------------------------------------
package appls.APPL1.xhtml.user;
import com.jacada.jis.runtime.server.frontend.xhtml.context.OnPageLoadContext;
import com.jacada.jis.runtime.server.frontend.xhtml.controls.*;
import com.jacada.jis.runtime.server.frontend.xhtml.general.JacadaControlTypes;
import java.util.Vector;
import java.util.Enumeration;
public class Appl {

    public void onPageLoad(OnPageLoadContext context) {
          Window window = context.getWindow();
```

```
   // you can either get the control by its name
          Date dateControl = (Date)window.getControlByName(“Date”);
          dateControl.setCalendarBackground(“red”);

   // Or you can get all the controls of a certain type
       Vector controls =  window.getControlsByType(JacadaControlTypes.DATE_CTRL);
   // Iterating through all the date control on page
   for(Enumeration e = controls.elements();e.hasMoreElements();){
          Date date = (Date)e.nextElement();
          date.setCalendarBackground(“green”);
       }
       }
}
//-------------------------------------------------------------------
```

Library Scope

If you want your extension code to affect all of the subapplications in a specific library, the extension file must be named `Appl.java`; the package name must be `appls.<LibName>.xhtml.user`

The runtime generation process generates the following two files automatically:

`Appl.java` - a skeleton file in which you insert your code

`jacc.bat` - a batch file used to compile the code in the appl.java file.

The above two files are generated in the directory:
`<InstallDir>\jacadafiles\src\appls\<LibName>\xhtml\user`

Use `jacc.bat` to compile the Java source code. The resulting classes are placed in the application classes directory:
`<InstallDir>\jacadafiles\classes\appls\<LibName>\xhtml\user`

For a library named LIB1, in an Application named APPL1, this is the code in the skeleton `Appl.java` file:

```
//----------------------------------------------------------------
package appls.LIB1.xhtml.user;
import com.jacada.jis.runtime.server.frontend.xhtml.extension.*;
import com.jacada.jis.runtime.server.frontend.xhtml.context.OnPageLoadContext;
import com.jacada.jis.runtime.server.frontend.xhtml.controls.*;
import com.jacada.jis.runtime.server.frontend.xhtml.general.JacadaControlTypes;
import java.util.Vector;
import java.util.Enumeration;
public class Appl extends app.<ApplName>.xhtml.user.Appl {
    public void onPageLoad(OnPageLoadContext context) {
          Window window = context.getWindow();
   // you can either get the control by its name
          Date dateControl = (Date)window.getControlByName(“Date”);
```

```
            dateControl.setCalendarBackground(“red”);
        // Or you can get all the controls of a certain type
            Vector controls =  window.getControlsByType(JacadaControlTypes.DATE_CTRL);
        // Iterating through all the date control on page
        for(Enumeration e = controls.elements();e.hasMoreElements();){
            Date date = (Date)e.nextElement();
            date.setCalendarBackground(“green”);
          }
          }
}
//------------------------------------------------------------------
```

Subapplication Scope

For code that affects a specific Subapplication only, the extension file must be named according to the Subapplication name: `<SubApplName>.java.`

For example, for a Subapplication named `SUBAPP1`, the name of the file is `SUBAPP1.java`.

In contrast to the skeleton files created automatically for Application and library extensions, a Subapplication extension file must be created by the user. The location and contents of the Subapplication file depend on whether the Subapplication resides in the main Application or in a library.

Subapplications Residing in a Library

When a Subapplication resides in a library, the `<SubApplName>.java` file must be placed in the library source code directory, together with the library’s `Appl.java` file:
`<InstallDir>\jacadafiles\`**`src`**`\appls\<`**`LibName`**`>\xhtml\user`

After using `jacc.bat` to compile the Java source code, the classes are placed in the library classes directory:
`<InstallDir>\jacadafiles\`**`classes`**`\appls\<`**`LibName`**`>\xhtml\user`

The Subapplication class should extend the library class. In such a case, the **`onPageLoad`** and **`onPageSubmit`** methods may call their super methods, as defined in the library extension.

Subapplications Residing in an Application

When a Subapplication resides in an Application, the `<SubApplName>.java` file must be placed in the Application source code directory, together with the Application’s `Appl.java` file:
`<InstallDir>\jacadafiles\`**`src`**`\appls\<`**`ApplName`**`>\xhtml\user`

After using `jacc.bat` to compile the Java source code, the classes are placed in the Application classes directory:
`<InstallDir>\jacadafiles\`**`classes`**`\appls\<`**`ApplName`**`>\xhtml\user`

The Subapplication class should extend the Application class. In such a case, the **onPageLoad** and **`onPageSubmit`** methods may call their super methods, as defined in the Application extension.

Subapplication SA1 is in Application APPL1. To create a Subapplication scope extension, create a file named `SA1.java` and use the skeleton below:

```
//----------------------------------------------------------------
package appls.APPL1.xhtml.user;
import com.jacada.jis.runtime.server.frontend.xhtml.extension.*;
import com.jacada.jis.runtime.server.frontend.xhtml.context.*;
import com.jacada.jis.runtime.server.frontend.xhtml.controls.*;
import com.jacada.jis.runtime.server.frontend.xhtml.general.JacadaControlTypes;
import java.util.Vector;
import java.util.Enumeration;
public class SA1 extends Appl {
Window window = context.getWindow();
   // you can either get the control by its name
          Date dateControl =
         (Date)window.getControlByName("Date");
          dateControl.setCalendarBackground("red");
   // Or you can get all the controls of a certain type
       Vector controls =  window.getControlsByType(JacadaControlTypes.DATE_CTRL);
   // Iterating through all the date control on page
   for(Enumeration e = controls.elements();e.hasMoreElements();){
          Date date = (Date)e.nextElement();
          date.setCalendarBackground("green");
       }
       }
}
//-----------------------------------------------------------------------------
public void onPageLoad(OnPageLoadContext context) {
   ... // Subapplication specific code
   super.onPageLoad(context);
}
public void onPageSubmit(OnPageSubmitContext onSubmitContext) {
   ... // Subapplication specific code
   super.onPageSubmit(onSubmitContext);
}
}
//-----------------------------------------------------------------------------
```

Note: When the Subapplication resides in a library, all you need to do is replace the Application name in the first line with the library name.

Modifying the Date Calendar Window in a Skin

Setting up an Application to use skins involves two main steps:

- Writing the control-level extension files
- Configuring the Application INI file

Note: You can also set and remove skins via DoMethods in ACE. For more information, see “Skin-related DoMethods” on page 292.

Writing Control-level Extension Files

Control-level extensions must be written in files with specific names, according to the control type.

The following file name must be used for the java code that is to modify the Date Calendar Window: `DateExtension.java`

For the file names to use for extensions for other controls, see “Writing Control-level Extension Files” on page 320.

The location of the control-level extension files can be determined by the user, as long as it is in the server classpath.

To modify the classpath, use the following guidelines:

- Development environment - anywhere under the following directory: `<InstallDir>\JacadaFiles\`
- Deployment environment - anywhere under the runtime installation directory.

Here is an example of a java extension that modifies the Date Calendar Window. The extension is for use in a skin.

```
//----------------------------------------------------------------------
package skins.Green;
import com.jacada.jis.runtime.server.frontend.xhtml.extension.
IControlExtension;
import com.jacada.jis.runtime.server.frontend.xhtml.context.
OnControlReadyContext;
import com.jacada.jis.runtime.server.frontend.xhtml.controls.Date;

public class DateExtension implements IControlExtension {
    public void onControlReady(OnControlReadyContext context) {
         date.setCalendarBackgroundColor(“orange”);
         date.setCalendarTitle(“mytitle”);
       }
}
//----------------------------------------------------------------------
```

You would compile the above file with the jacc.bat procedure. The output class will go to `<InstallDir>\JacadaFiles\classes\skins\Green`

Configuring the Application INI File

To use control-level extensions, you need to configure the Application `.ini` file.

The Application INI file is named according to the Application, and is located in the following directory of the development environment:
`<InstallDir>\appls\<ApplName>\rt32\<ApplName>.INI`

To configure the Application INI file to use control-level extensions, or skins, follow these steps:

- Add a new section - `[Skins]`
- Within the `[Skins]` section, define all the skins to be used in this Application, as follows:

  ```
  skinName1=<skin1 packagename>
  skinName2=<skin2 packagename>
  ```

  For example:

  ```
  skinName1=skins.red
  skinName2=skins.green
  ```

  The package specified must correspond to the location of the extension files.
- Specify the active skin, as follows:

  ```
  Active = skinName1
  ```

  In order for the extensions in the package to be activated when the Application is loaded, the active skin name value must be a valid name from the list. Otherwise, none of the extensions are activated.

Example 16. Use of skins- <ApplName>.ini settings

The runtime installation of Application A1 is at c:\JIS. The directory c:\JIS\skins\ contains two subdirectories, red and green, in which extension files were placed. These extension files contain the properties of each one of the skins. The current active skin is green. Below is an example of the `[Skins]` section in the `A1.ini` file.

```
[Skins]
red = skins.red
green = skins.green
Active = green
```

Additional APIs

This section contains additional APIs for modifying controls.

Combo Box Methods

The following methods are available in the `GUIComboBox` class, for working with the Combo Box control.

addItem

```
public void addItem(java.lang.String newItemStr)
```

Adds an item to a Combo Box.

Parameters:

`newItemStr` - the item to add to the Combo Box

addItem

```
public void addItem(java.lang.String item,
                    int index)
```

Adds an item to a Combo Box by index value.

Parameters:

`item` - the text string to add to the Combo Box

`index` - the position the item will be inserted, zero based.

cloneControl

```
public ComboBox cloneControl(java.lang.String newName)
```

Clones a Combo Box.

Parameters:

`newName` - the name of the new control

Returns:

a new Combo Box control

deleteAllItems

`public void` **`deleteAllItems`**`()`

Remove all the items in the Combo Box

deleteItem

`public void` **`deleteItem`**`(int index)`

Remove an item from the Combo Box

Parameters:

`index` - the index of the item to be removed

deleteItem

`public void` **`deleteItem`**`(java.lang.String item)`

Deletes an item from a Combo Box.

Parameters:

`item` - the item to remove from the Combo Box

getItem

`public java.lang.String` **`getItem`**`(int index)`

Get an item from the Combo Box.

Parameters:

`index` - the index of the item in the combo box items list - zero based

Returns:

the items string value

getItemCount

`public int` **`getItemCount`**`()`

Get the number of items in the Combo Box.

Returns:

the number of items in the Combo Box.

getSelectedIndex

`public int` **`getSelectedIndex`**`()`

Get the number of items in the Combo Box.

Returns:

the index of the selected item or -1.

getSelectedItem

`public java.lang.String` **`getSelectedItem`**`()`

Returns the selected item in a Combo Box.

Returns:

the selected item in the Combo Box

See Also:

getText

getText

`public java.lang.String` **`getText`**`()`

Returns the selected item in a Combo Box.

Overrides:

`getText` in class `XhtmlControl`

Returns:

the selected item in the Combo Box

See Also:

getSelectedItem

setSelectedItem

`public void` **`setSelectedItem`**`(int index)`

Select an item by its index.

Parameters:

`index` - the index of the item to be selected

setSelectedItem

`public void` **`setSelectedItem`**`(java.lang.String text)`

Sets the selected item in a Combo Box.

Parameters:

`text` - the item to select in the Combo Box

See Also:

setText

setText

`public void` **`setText`**`(java.lang.String text)`

Sets the selected item in a Combo Box.

Overrides:

`setText` in class `XhtmlControl`

Parameters:

`text` - the item to select in the Combo Box

See Also:

setSelectedItem

Chapter 10. Conducting XML-based Transactions from the Client

It is possible to grant end-users partial controlled access to your host application, for the purpose of conducting B2B transactions. End-users can submit a URL request with an attached XML object, and receive an answer from the host, also in the form of an XML object. This is known as an XML Transaction. In this case, end-users communicate with the host, but are not privy to the host's inner workings.

As far as end-users are concerned, such communication is usually in the form of a structured query, which is part of a larger web application's HTML form. The end-users merely submit this HTML form and receive a reply to their query. During this time, the end-users are completely oblivious of the XML Transactions and the real-time processing of these transactions by the JIS Server.

The following topics are discussed:

- "XML-Based Transactions" on page 327
- "XML-based DoMethods" on page 330
- "Navigating Through Host Screens" on page 350

XML-Based Transactions

This chapter discusses XML-based Transactions, how XML-based Transactions work, how to construct XML-based Transaction, and how to merge XML Transactions with an existing XHTML Client Application.

Chapter 11 - "The Server Configuration File" on page 359 also addresses XML-based Transactions.

The Transaction Dataflow

The following sketch outlines a typical transaction's dataflow:

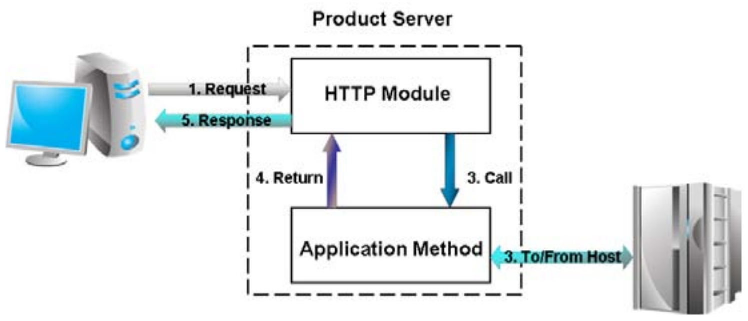


Figure 72. A typical XML transaction

The Transaction Dataflow:

1 The end-user sends a URL request via HTTP to the JIS Server with an attached XML object.
2 The JIS Server:
   - Receives and analyzes the request.
   - Opens a host session.
   - Invokes a transaction method.
3 A transaction method performs the following:
   - Parses the attached XML object.
   - Navigates through the host application's screens.
   - Retrieves information from the host

   -OR-
   - Writes information to the host.
4 The JIS Server creates an XML object, which contains the requested information.
5 The JIS Server sends a response back to the end-user with the attached XML object.

Assembling a Transaction

Once a host session is established, you can execute a transaction. To assemble a transaction, you must carefully plan your steps, according to the transaction workflow.

You enable the use of transactions by doing the following:

- Create transaction methods in ACE.
- Create (or modify) the Server Configuration File.

Transaction Methods

A transaction method is a User-Triggered method that is triggered upon the end-user's request. When the end-user requests a transaction, the JIS Server analyzes the request according to the Server Configuration File's settings and invokes a transaction method.

A typical transaction method does the following:

1. Receives an XML document from the end-user through an input stream.
2. Parses the XML document.
3. Logs into the host application.
4. Navigates through the host screens to the appropriate screen.
5. Either:
   - Retrieves the desired information from host screen fields.

   -or-

   - Inserts information into host screen fields.
6. Writes a new XML document containing the reply.
7. Sends an XML document to the end-user through an output stream.

All of these steps are governed by ACE methods.

Note: The order above is merely a recommendation. You can change the order to fit your needs. For example, you can perform step 3 before step 2.

XML-based DoMethods

To write transaction methods, you must be familiar with XML-based DoMethods. This section lists these DoMethods and explains how to use them.

webMethods JIS provides two ways of implementing XML:

- The JIS DOM-based API
- The JIS template API

The process of reading, parsing and creating XML documents is outlined separately for these two implementations.

The DOM-based API

DOM (Document Object Model), a programming interface specification being developed by the World Wide Web Consortium (W3C), lets a programmer create and modify HTML pages and XML documents as full-fledged program objects. Currently, HTML and XML are ways to express a document in terms of data structure. DOM is a strategic and open effort to specify how to provide programming control over documents.

You can create XML documents based on this document object model by using the JIS DOM-based API.

Since the JIS Server is written in Java, the interface resembles the Java Sun API for XML. This API contains quite a few objects with a few dozens of methods. The set of objects and functions is minimal, allowing you to perform simple tasks like creating and manipulating an XML file.

The DoMethods in the DOM-based API use the following objects as receivers:

- XMLEngine
- XMLTreeDocument
- XMLElementList
- XMLElement
- ExternalData (partially)

DoMethods Used with the DOM-based API

The following tables explain the DoMethods most commonly used with the DOM-based API. In the end of this chapter is a full list of XML DoMethods.

DoMethod Name	GetExternalInputStream
Description	Returns the input stream object containing the request data
Receiver	ExternalData
Return type and value	Returns a JSInputStream.
Parameters	This DoMethod has no parameters.

DoMethod Name	ReadXMLTreeDocument
Description	Reads and builds a DOM-based XML document object from the input stream.
Receiver	XMLEngine
Return type and value	Returns an XMLTreeDocument.
Parameters	inStream - Enter the Input Stream object, the outcome of the **`GetExternalInputStream`** DoMethod. validate - Specify whether to perform validation on the document that was read (boolean). This parameter is applicable only if the XML document contains a reference to a DTD.

DoMethod Name	GetDocumentElement
Description	Returns the root element of a document.

Receiver	XMLTreeDocument - Apply this DoMethod to the outcome of the **`ReadXMLTreeDocument`** DoMethod.
Return type and value	Returns the root element of a document which is an XMLElement.
Parameters	This DoMethod has no parameters.

DoMethod Name	GetElementsByTagName
Description	Returns a list of nodes that have the same tag name from the document. Useful for data manipulations.
Receiver	XMLTreeDocument - Apply this DoMethod to the outcome of the **`ReadXMLTreeDocument`** DoMethod.
Return type and value	Returns an XMLElementList.
Parameters	name - Enter the tag name.

DoMethod Name	GetChildNodes
Description	Returns a list of this node's child elements. Useful for searching within a node.
Receiver	XMLElement - Apply this DoMethod to the outcome of the **`GetDocumentElement`** DoMethod or to a specific node.
Return type and value	Returns an XMLElementList.
Parameters	This DoMethod has no parameters.

DoMethod Name	GetItem or Item
Description	Returns the item at a specified position index in the node list.
Receiver	XMLElementList - Apply this DoMethod to the outcome of the **`GetChildNodes`** DoMethod or the **`GetElementsByTagName`** DoMethod.
Return type and value	Returns an XMLElement.
Parameters	index - Enter the index number of the element's position in the list. The list may include only one element, or possibly even no elements whatsoever.

DoMethod Name	GetTagName
Description	Gets this node's tag name.
Receiver	XMLElement - Apply this DoMethod to a specific node.
Return type and value	String
Parameters	This DoMethod has no parameters.

DoMethod Name	GetNodeTextValue
Description	Gets the text value of a node. *Example*: `<last_name>Melville</last_name>` The method returns "Melville".
Receiver	XMLElement - Apply this DoMethod to a specific node.
Return type and value	String
Parameters	This DoMethod has no parameter

DoMethod Name	GetAttribute
Description	Gets a value of an attribute of this node *Example*: `<book ISBN="465467" >` The method gets "ISBN" and returns "465467".
Receiver	XMLElement - Apply this DoMethod to a specific node.
Return type and value	String
Parameters	attributeName - Enter the attribute name.

DoMethod Name	NewXMLTreeDocument
Description	Creates a new empty XMLTreeDocument object.
Receiver	XMLEngine
Return type and value	XMLTreeDocument
Parameters	This DoMethod has no parameters.

DoMethod Name	CreateElement
Description	Creates a new element and provides it with a tag name.
Receiver	XMLTreeDocument - Apply this DoMethod to the outcome of the **`NewXMLTreeDocument`** DoMethod.
Return type and value	XMLElement
Parameters	tagName - Enter the node tag name.

DoMethod Name	AppendChild
Description	Appends an element as a child node of another element. When appending to a NewXMLTreeDocument, the node becomes the root element.
Receiver	XMLElement or NewXMLTreeDocument
Return type and value	Boolean
Parameters	node - Enter the child node object or the root node object.

DoMethod Name	SetAttribute
Description	Adds an attribute to a node and sets its value.
Receiver	XMLElement
Return type and value	Boolean
Parameters	name - Enter the attribute name. value - Enter the attribute value.

DoMethod Name	AppendTextChild
Description	Inserts the text specified in the parameter into the element.
Receiver	XMLElement - Apply this method to a specific node.
Return type and value	Boolean
Parameters	text - Enter the node's text.

DoMethod Name	GetExternalOutputStream
Description	Returns an output stream object in order to send data from the JIS Server to the end-user.
Receiver	ExternalData
Return type and value	JSOutputStream
Parameters	This DoMethod has no parameters.

DoMethod Name	WriteXMLTreeDocument
Description	Writes a DOM-based XML document to an output stream.
Receiver	XMLEngine
Return type and value	Boolean
Parameters	doc - Enter the DOM-based XML document object. outStream - Enter the Output Stream object.

Parsing XML Documents Using the DOM API

To read an XML document, you parse it and retrieve its parameters' values.

The following DoMethods are used for each step of the process.

1 Establish an input stream using the **`GetExternalInputStream`** DoMethod.

2 Build an XMLTreeDocument from the input stream using the **`ReadXMLTreeDocument`** DoMethod and referencing it to the outcome of the **`GetExternalInputStream`** method line.

3 Identify the XML object's root element using the **`GetDocumentElement`** DoMethod.

4 Search within the XML object's nodes, by doing one of the following:

- Extract a list of nodes with a specific name, using the **`GetElementsByTagName`** DoMethod. This list may contain only one element, when only one element with this name exists in the document. The list may even be empty, if there is no element with this name in the document.

-OR-

- Search the tree for a node with a specific characteristic such as a specific attribute, using the **`GetChildNodes`** DoMethod to produce a list, and then search through it, element by element.

5 Pick an element from the list using the **`GetItem`** DoMethod or the **`Item`** DoMethod.

6 Use the **`FileOpenRead`** DoMethod instead of using an HTTP stream to open an XML file and read the information from it.

7 Use the **`ReadToString`** DoMethod to transform the InputStream into a string.

Retrieving the Data Using the DOM API

After finding the desired XML element, you will probably need to retrieve the element value or attribute.

To retrieve an element's name, value or attribute:

1 Retrieve a tag's name using the **`GetTagName`** DoMethod.
2 Retrieve a tag's value using the **`GetNodeTextValue`** DoMethod.
3 Retrieve the value of a tag's attribute using the **`GetAttribute`** DoMethod.

Construction of XML Documents Using the DOM API

When you are ready to pack all the data that you collected into an XML file, you must create a new XML document object.

To construct an XML document:

1 Create a new XML document, using the **`NewXmlTreeDocument`** DoMethod.
2 Create the root node, by applying the **`CreateElement`** DoMethod to the outcome of the **`NewXMLTreeDocument`** method line.
3 Append the root node to the XML tree, by referencing the **`AppendChild`** DoMethod to the **`NewXmlTreeDocument`** method line.
4 Create a new node using the **`CreateElement`** DoMethod.
5 Give the newly created node an attribute, if needed, using the **`SetAttribute`** DoMethod.
6 Give the newly created node a tag value, if needed, using the **`AppendTextChild`** DoMethod.
7 Subject the newly created node to its parent node, referencing the **`AppendChild`** DoMethod to the parent node's **`CreateElement`** method line.
8 Repeat steps 4-7 for every additional element, as needed.

Writing an XML Tree Document and Sending it Back to the End-user

In order to complete the transaction you will need to send a response back to the end-user who initiated the transaction.

To write an XML document and send it back to the end-user:

1 Establish output stream, using the **`GetExternalOutputStream`** DoMethod.
2 Transmit your newly created XML document through the output stream, using the **`WriteXMLTreeDocument`** DoMethod.

3 Use the **`FileOpenWrite`** DoMethod instead of using an HTTP stream to open an XML file and write the information to it.

4 You can use the **`PrintString`** DoMethod to write a simple string into the OutputStream object.

Note: You can close Input/Output streams using the **`Close`** DoMethod. However, even if you do not close the streams, they will close automatically upon method's completion.

The JIS Template API

A JIS Template is an XML file that you create with a .tpl suffix. JIS Templates simplify work by prescribing the structure of the XML files. By working with a template, you are able to determine what the XML document should look like. A template contains JIS attributes and is structured to fit the transaction's requirements.

To construct a new XML document according to an existing JIS template, you use the ACE methods discussed in this section, referring to nodes by their JIS names. In order to add new nodes, the developer must duplicate existing nodes in the template and change the tags and attribute values using special methods.

The JIS names in the template file are unique and should appear only once as a node's identifier.

Example of a template file:

```
<Receipt>
   <Item Type="Type" Name="Name" JISName="Item">
      <Price JISName="Price">Value</Price>
      <Quantity JISName="Quantity">Value</Quantity>
      <Comment JISName="Comment">Value</Comment>
   </Item>
</Receipt>
```

Note: This file contains one node named Item. The <Item> node contains attributes and child nodes. Each node has a JIS name. The JIS name enables you to refer to this node directly, without navigating to it. The JISName attribute does not have to be the same as the tag name. You can change the values of the nodes, or leave them as they are. You can add a node by duplicating an existing node.

DoMethods Used With the JIS Template API

The following tables explain the DoMethods most commonly used with the JIS Template API. A full list of XML DoMethods can be found in Table 65.

DoMethod Name	GetExternalInputStream
Description	Returns the input stream object containing the data from the request.
Receiver	ExternalData
Return type and value	Returns a JSInputStream.
Parameters	This DoMethod uses no parameters.

DoMethod Name	ParseDocumentByTemplate
Description	Parses an XML object read from the input stream, according to the specified template, and inserts JISNames into the nodes.
Receiver	XMLEngine
Return type and value	XMLTemplateDocument
Parameters	inStream - Enter the Input Stream object. templateName - Enter the template file's path and name. validate - Enter whether or not to perform validation on the input document.

DoMethod Name	MatchNextNode
Description	When parsing the document, using the **`ParseDocumentByTemplate`** DoMethod, only the first node matching each JISName is returned. If there is more then one node in the document that matches the same JISName, use this DoMethod to move to the next node that matches the same JISName. Once another match is found, it is not possible to return to the previous one using this DoMethod.
Receiver	XMLTemplateDocument
Return type and value	Boolean
Parameters	nodeJISName - Enter the node's JIS name.

DoMethod Name	GetNodeValue
Description	Gets the text value of a node. *Example*: <last_name>Melville</last_name> The method returns "Melville".
Receiver	XMLTemplateDocument.
Return type and value	String
Parameters	nodeJISName - Enter the node's JIS name.

DoMethod Name	GetNodeAttribute
Description	Gets a value of an attribute of this node. *Example*: <book ISBN="465467" > The method gets "ISBN" and returns "465467".
Receiver	XMLElement - Apply this method to a specific node.
Return type and value	String
Parameters	attributeName - Enter the attribute name. nodeJISName - Enter the JIS node name.

DoMethod Name	NewDocumentFromTemplate
Description	Builds a new XML document based on the template specified in the parameter.
Receiver	XMLEngine
Return type and value	XMLTemplateDocument
Parameters	templateName - Enter the template file’s path and name.

DoMethod Name	DuplicateNode
Description	Adds a sibling node to the node specified in the parameter according to its JISName.
Receiver	XMLTemplateDocument
Return type and value	Boolean
Parameters	nodeJISName - Enter the JIS node name.

DoMethod Name	RemoveNode
Description	Removes a node and all its child nodes from a document.
Receiver	XMLTemplateDocument
Return type and value	Boolean
Parameters	nodeJISName - Enter the JIS node name.

DoMethod Name	SetNodeValue
Description	Inserts the text into the node. The node is identified by its JISName.
Receiver	XMLTreeDocument
Return type and value	Boolean
Parameters	nodeJISName - Enter the JIS node name. str - Enter the text of the node.

DoMethod Name	SetNodeAttribute
Description	Sets the value of a node attribute If the attribute already has a value, it is replaced. The node is identified by its JISName.
Receiver	XMLTreeDocument
Return type and value	Boolean
Parameters	nodeJISName - Enter the JIS node name. attributeName - Enter the attribute name. value - Enter the attribute value.

DoMethod Name	GetExternalOutputStream
Description	Returns an output stream object in order to send data from the server to the end-user.
Receiver	ExternalData
Return type and value	JSOutputStream
Parameters	This DoMethod uses no parameters.

DoMethod Name	WriteXMLTemplateDocument
Description	Writes a Template-based XML document to an output stream.
Receiver	XMLEngine
Return type and value	Boolean
Parameters	doc - Enter the Template-based XML document object. outStream - Enter the Output Stream object.

Parsing Documents Using the Template API

The template applied to the document is read from a stream. You must make sure that the document being parsed has the same structure as the template. Otherwise, it will not be parsed. If the template cannot be applied, the function returns a return code indicating an error.

To parse the XML document:

1. Establish an input stream using the **`GetExternalInputStream`** DoMethod.
2. Use the **`ParseDocumentByTemplate`** DoMethod and reference it to the **`GetExternalInputStream`** method line.

   The **`ParseDocumentByTemplate`** DoMethod returns the first node that matches each JISName.
3. Get to the next node with a matching JISName, using the **`MatchNextNode`** DoMethod.

Retrieving the Data of a Template-based Document

After parsing a template-based document, you will want to retrieve its elements' values.

To retrieve an element's value or attribute:

1. Retrieve a tag's value, using the **`GetNodeValue`** DoMethod.
2. Retrieve the value of a tag's attribute, use the **`GetNodeAttribute`** DoMethod.

Note: Remember to navigate to the desired node using the **`MatchNextNode`** DoMethod, before retrieving its values.

Constructing New XML Documents Using the Template API

You can use a template-based document as a response to the end-user.

To construct a template-based XML document:

1. Create a new XML document based on a template using the **`NewDocumentFromTemplate`** DoMethod.
2. Duplicate nodes from the template as needed using the **`DuplicateNode`** DoMethod.
3. Remove unnecessary nodes using the **`RemoveNode`** DoMethod.
4. Change the values of nodes and their attributes as needed by doing the following:
   - Set an element's **value**, using the **`SetNodeValue`** DoMethod.
   - Set an element's **attribute**, using the **`SetNodeAttribute`** DoMethod.

Note: Use existing nodes from the template. Duplicate them or delete the ones you do not need.

Writing an XML Template-based Document and Sending it Back to the End-user

Once you have finished creating the document, you must send a response back to the end-user who requested the transaction.

To send a template-based XML file to the end-user:

1. Establish an output stream using the **`GetExternalOutputStream`** DoMethod.
2. Using the **`WriteXMLTemplateDocument`** DoMethod, send the XML file to the end-user through the output stream.

Note: You can close the Input/Output streams using the **`Close`** DoMethod. However, even if do not do close the streams, they will close automatically upon the method's completion

List of XML-based DoMethods

Table 65 presents a full list of XML-based DoMethods.

Table 65. XML-based DoMethods (Sheet 1 of 4)

DoMethod Name	Description	Return type
`AppendChild`	Appends an element as child node to other elements. When appended to a NewXMLTreeDocument, the node becomes the root element. Use in DOM-based XML methods.	Boolean
`AppendTextChild`	Inserts the text, specified in the parameter, into the element. Use in DOM-based XML methods.	Boolean
`Close`	Closes an InputStream or an OutputStream.	Void
`CreateElement`	Creates a new XML element. Use in DOM-based XML methods.	XML Element
`DuplicateNode`	Adds a brother node to the node specified in the parameter by its JIS name. Use in template-based XML methods.	Boolean
`GetAttribute`	Returns the value of a node's attribute. The attribute's name is specified in the parameter. Use in DOM-based XML methods.	String
`GetChildNodes`	Returns a list of the child XML elements of a node. Use in DOM-based XML methods.	XML Element List
`GetDocumentElement`	Returns the root element of an XML document. Use in DOM-based XML methods.	XML Element

Table 65. XML-based DoMethods (Sheet 2 of 4)

DoMethod Name	Description	Return type
`GetElementsBy TagName`	Returns a list of the elements named according to name used in the parameter. Use in DOM-based XML methods.	XML Element List
`GetExternalInput Stream`	Returns an InputStream object from an external data source.	JSInput Stream
`GetExternalOutput Stream`	Returns an output stream object in order to send data from the server to the end-user.	JSOutput Stream
`GetItem`	Returns an XML element from a list. The element's index in the list is specified in the parameter. Use in DOM-based XML methods.	XML Element
`GetLength`	Returns the number of elements contained in a list of elements. Use in DOM-based XML methods.	Int
`GetNodeAttribute`	Returns the value of a node's attribute. The attribute is specified in a parameter. Use in a template-based XML method.	String
`GetNodeTextValue`	Returns the text found in a node. Use in DOM-based XML methods.	String
`GetNodeValue`	Returns the text found in a node. Use in template-based XML methods.	String
`GetTagName`	Returns the name of an element's tag. Use in DOM-based XML methods.	String
`GetXMLElement ByName`	Returns a DOM-based element from a node in a template-based XML document. This DoMethod connects the JIS template interface with the DOM interface.	XML Element

Table 65. XML-based DoMethods (Sheet 3 of 4)

DoMethod Name	Description	Return type
`GetXmlLastError`	Returns a description of the last error to occur during the execution of the method. Use in DOM-based or template-based XML methods.	String
`Item`	Returns an XML element from a list. The element's index in the list is specified in the parameter. Use in DOM-based XML methods.	XML Element
`MatchNextNode`	Moves to the next node that has the same JISName, as specified in the parameter. Use in template-based XML methods.	Boolean
`NewDocumentFrom Template`	Builds a new XML document based on the template specified in the parameter. Use in template-based XML methods.	XML Template Document
`NewXMLTreeDocument`	Creates a new empty XML document. Use in DOM-based XML methods.	XMLTree Document
`ParseDocument ByTemplate`	Parses an XML object read from the input stream, according to the specified template, and inserts JISNames into the nodes. Use in template-based XML methods.	XML Template Document
`PrintString`	Prints a string into the OutputStream.	Void
`ReadToString`	Transforms an InputStream into a string. This function empties the input stream.	String
`RemoveNode`	Removes a node and all its child nodes from a document. Use in template-based XML methods.	Boolean

Table 65. XML-based DoMethods (Sheet 4 of 4)

DoMethod Name	Description	Return type
`SetAttribute`	Adds an attribute to a node and sets its value. If the attribute already exists, its value is changed to the new value. Use in DOM-based XML methods.	Boolean
`SetNodeAttribute`	Sets the value of a node's attribute. If the attribute already has a value, it is replaced by the new value. Use in a template-based XML method.	Boolean
`SetNodeValue`	Inserts the text specified in the parameter into a node. The node is identified by its JISName. Use in template-based XML methods.	Boolean
`ToXMLTreeDocument`	Converts a template-based XML document into a DOM-based XML document.	XMLTree Document
`WriteXMLTemplate Document`	Writes a template-based XML document to the output stream. Use in template-based XML methods.	Boolean
`WriteXMLTree Document`	Writes an XML document based on DOM syntax to the output stream. Use in DOM-based XML methods.	Boolean

Navigating Through Host Screens

Once a session is allocated, the JIS runtime Application navigates through the host screens to the desired screen, in which information is inserted or extracted, according to the transaction's specifications.

Navigation is accomplished by logging into the host application and skipping screens.

Navigation Types

There are two ways to navigate through the host screens:

- Single-method navigation
- Multiple-method navigation

Both are acceptable. However, we recommend multiple-method navigation.

Single-method Navigation

Single-method navigation is accomplished by performing all of the navigation within one method.

To perform single-method navigation:

1. In the transaction method, after parsing the XML object (if any), log into the host application.
2. For the next screen, fill the appropriate field to move to the next screen. Usually you will have to insert a number in the menu command field, press the **Enter** key and move-according-to-host.
3. Repeat step 2 for every new screen, until you reach the desired screen.
4. For the desired screen, use the appropriate DoMethods to write information to the desired fields and/or retrieve information from the desired fields.

Advantage of Single-method Navigation **Simple**. You can view the whole transaction at once in a linear order.

Disadvantage of Single-method Navigation **Bulky.** The method is very long, especially with large host applications requiring navigation through many screens.

Multiple-method Navigation

Multiple-method navigation is accomplished by performing the navigation with many methods. This is done either by calling more methods from the transaction method, or by modifying the **`UserInitSubApplication`** System-Triggered method of every Subapplication in the course of the navigation.

To perform multiple-method navigation:

1. In the transaction method, after parsing the XML object (if any), log into the host application. This takes about three method lines.
2. Do one of the following:
   - Call another method, by inserting a DoMethod method line and choosing a method from the DoMethod list. This new method can be a method that

you have prepared in ACE to perform all or part of the navigation to the desired screen.

-OR-

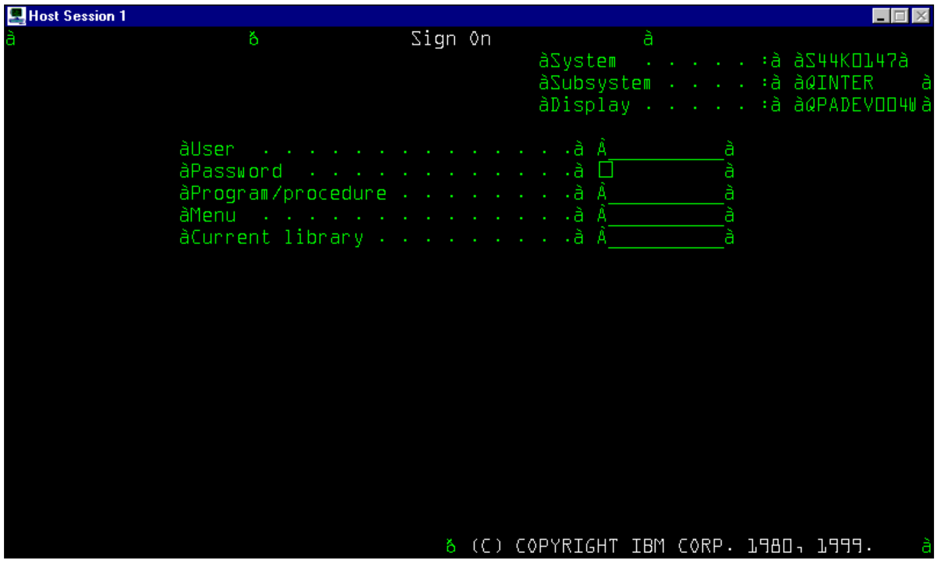
- In the next screen's Subapplication, modify the **`UserInitSubApplication`** System-Triggered method so that it moves to the next screen. You can repeat this for every screen until you reach the desired screen.

Advantage of Multiple-method Navigation **Short.** Every method is short.

Disadvantage of Multiple-method Navigation **Complex**. Many intertwined methods make it difficult to comprehend the navigation process.

Example: Navigating Through a SignOn Screen

In the following example, the JIS Server has allocated a session and the first host screen the runtime identifies is a SignOn screen.



```
Host Session 1
à                   ŏ            Sign On            à
                                           àSystem  . . . . . :à àS44K0147à
                                           àSubsystem . . . . :à àQINTER    à
                                           àDisplay . . . . . :à àQPADEV004W à

              àUser  . . . . . . . . . . . . . .à Â__________à
              àPassword  . . . . . . . . . . . .à □          à
              àProgram/procedure . . . . . . . .à Â__________à
              àMenu  . . . . . . . . . . . . . .à Â__________à
              àCurrent library . . . . . . . . .à Â__________à

                                    ŏ (C) COPYRIGHT IBM CORP. 1980, 1999.    à
```

Figure 73. SignOn screen

The method must do the following on the host:

1 Fill in the "UserName" and "Password" fields.

2 Press the **Enter** key.

3 Move according to host.

These are performed using the following method lines:

```
HostType: Text: `"JOSEPH"` Field: User MoveCursor: False
AidKey: AidNone RemainInScreen: False
```

```
HostType: Text: `"JO77"` Field: Password MoveCursor: False
AidKey: AidEnter RemainInScreen: False
DoMethod: Receiver: `SubApplication` Method: MoveAccordingToHost Parms: (  )
```

The first method line types "JOSEPH" in the User field. The second method line types "JO77" in the Password field and presses **Enter**. The third method line moves according to host.

Retrieving Information from Host Screen Fields

You retrieve information from host screen fields and/or table cells using a series of **`Get`** DoMethods. DoMethods used for retrieving information from the host:

- **`GetDataFromScreen`**
- **`GetString`**
- **`GetVarValueByName`**
- **`GetUserVariable`**
- **`GetTableVar`**
- **`GetTopIndex`**
- **`GetStringByColumnIndex`**
- **`GetStringByColumnName`**

Writing Information into Host Screen Fields

You write information into host screen fields by inserting a HostType method line. You can also select a field or table cell and then modify its contents. DoMethods used for selecting and modifying host screen fields:

- **`SetCurTableSel`**
- **`MoveCursorToFieldByName`**
- **`MoveCursorToRowCol`**
- **`Add`**
- **`ClearHostFieldByName`**

Skipping Windows

To effectively navigate through host screens, a Subapplication can be processed in runtime without displaying its window, as long as information can be received from the host, and sent to the host, without the end-user's intervention. Essentially the Subapplication's window-display is skipped over.

By utilizing the Skipping Windows feature you can improve the performance time of your converted Application. With less information to process and less GUIs to build, the Application works faster.

Windows can be configured to always be skipped or skipped only when certain conditions exist. You choose the methods that control the skipping process.

It is important to remember that when using this technique the host window is not skipped. Rather, only the GUI equivalent of a host window is skipped.

For complete details about the Skipping Windows feature including a description of the various methods used for Skipping Windows, see the chapter in *webMethods JIS: Advanced Topics*.

Navigation Continuity

This section discusses advanced host screen navigation topics. To provide your transactions with good navigation fluidity, you must carefully plan every stage of the host screen navigation. This way you will minimize the possibility of communication problems between the JIS Server and the host, during the working of a transaction.

This section discusses the following:

- Navigation Strategies
- Overcoming Navigation Obstacles

Navigation Strategies

You will probably want to enable the JIS Server to perform several different transactions, each requiring navigation to different host screens. This requires intricate strategic planning. Here we shall outline several strategies for doing this.

User-Triggered Screen Skipping

User-triggered screen skipping is defined using user-triggered methods. This can be done with one or more methods. Usually, this form of navigation requires more than one user-triggered method. However, in some cases you may prefer to define all of the navigation in a single method.

For example, when a transaction requires a very short navigation through no more than three host screens, it is preferred to define all of the navigation within one method. This can be accomplished by performing all of the navigation within the initial transaction method. In this way, you have all of the navigation and XML-based method lines in on method. This is also known as single-method navigation.

System-Triggered Screen Skipping

When a transaction must navigate through many host screens, a common way of accomplishing this is through the use of system-triggered screen skipping.

This is accomplished by doing the following:

1 In the transaction method, log into the host application and **`MoveAccordingToHost`**.
2 In the following screen's **`UserInitSubApplication`** or **`UserSkipScreen`** system-triggered method, fill in the appropriate host screen fields to skip to the next screen and **`MoveAccordingToHost`**.
3 Repeat step 2 in every new host screen until you reach the desired screen.

Common-Navigation Methods

When several transactions use a common navigation route until they reach an "intersection", from which they split to different directions, it is recommended that they all use a common-navigation method to take them through these common host screens.

Example of Common Navigation Methods' Use There are three transactions named A, B and C:

1 Transaction A navigates through all the host screens from screen 1 to screen 11.
2 Transaction B navigates through all the host screens from screen 1 to screen 7, and from screen 7 it jumps to host screen 12.
3 Transaction C navigates through all the host screens from screen 1 to screen 10, and from screen 10 it jumps to host screen 13.

These transactions share common navigation routes on the way to their destination screens as seen below:

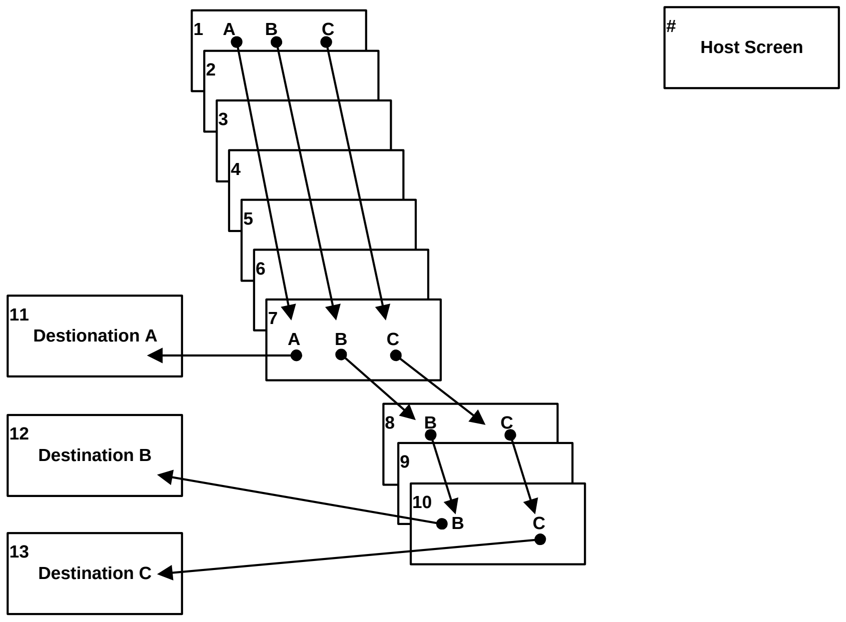


Figure 74. Navigation through several screens

The common navigation routes are as follows:

- Transactions A, B and C all navigate through host screens 1, 2, 3, 4, 5, 6 and 7.
- Transactions A and C both navigate through host screens 7, 8, 9 and 10.

This is where common-navigation methods come into place.

To make your work simpler, you can:

- Create a navigation method named **`NavigateOneToSeven`** for navigating from host screen 1 to host screen 7.
- Create a navigation method named **`NavigateSevenToTen`** for navigating from host screen 7 to host screen 10.

Transaction A In its transaction method, transaction A can now call for the **`NavigateOneToSeven`** method and then skip to host screen 11.

Transaction B In its transaction method, transaction B can now call for the **`NavigateOneToSeven`** method, then call for the **`NavigateSevenToTen`** method and then skip to host screen 12.

Transaction C In its transaction method, transaction C can now call for the **`NavigateOneToSeven`** method, then call for the **`NavigateSevenToTen`** method and then skip to host screen 13.

Note: This manner of navigating through host screens is recommended when your Application enables many transactions.

Overcoming Navigation Obstacles

A situation called “behind-the-scenes navigation” may occur on a mainframe computer, and also, less commonly, on an iSeries computer. This situation can sometimes lead to problems which ultimately cause the JIS Server to fail to recognize some screens, and even to stop running altogether. This ultimately causes the OutputStream to close before navigation is completed.

The navigation process:

1. The host starts navigating between screens, filling in and retrieving data from fields.
2. The host screen is updated by the host.
3. After a few moments, the completed output screen is sent back to the JIS Server.

In most cases, the JIS Server has no problem recognizing new or refreshed screens. Sometimes, however, the host sends an unexpected or an incomplete screen. Such screens are not recognized by the webMethods JIS runtime, and may block the runtime. As a result of this, the JIS Server may possibly produce a JITGUI.

To overcome this problem it is advised to use the Wait For Screen feature. See “Wait for Screen” on page 54

Chapter 11. The Server Configuration File

The Server Configuration File is an XML document that defines each and every transaction and its relation to the Client URL request, and provides the settings for the host sessions that house these transactions.

In order to successfully optimize the Server Configuration File, you should be familiar with the file's structure and its different element tags.

This document outlines the Server Configuration File's structure and provides guidelines for understanding the Server Configuration File and writing a new one.

The following topics are discussed:

- "How the Server Configuration File Works" on page 359
- "Getting Started" on page 362
- "Writing the Server Configuration File" on page 364
- "Enabling Action Definitions to be Overridden by the Client" on page 371
- "Enabling Session Definitions to be Overridden by Actions" on page 376
- "Example Server Configuration File" on page 381
- "XML Tag Reference" on page 387

How the Server Configuration File Works

The Server Configuration File regulates the flow of information between the Client application, the JIS application and the Host application.

Three-tier Hierarchy

The Server Configuration File is based on the principle of a three-tier hierarchy, in which one element overrides another.

The three elements of this hierarchy are:

Client	Requests a transaction and overrides the transaction's default settings.

Transaction	Designates a session and overrides the session's default settings.
Session	Connection to the host.

The Client

The Client sends a URL request for a specific transaction.

The parameters contained in this request override the default transaction settings. This means that the transaction's settings are subject to the Client's request.

The Transaction

When requested, a transaction accommodates itself within a session.

The implementation of a transaction is pre-defined within default transaction settings. These settings may change to accommodate the Client's request. The transaction's parameters override the default session settings. This means that the session settings are subject to the transaction it "houses".

A pre-defined transaction is known as an **Action Definition**.

The Session

When initiated, a session is opened on the host.

The opening, maintaining and freeing of a host session, are pre-defined within default session settings. These settings may change to accommodate the transaction. A pre-defined session is known as a **Session Definition**.

Overriding Dataflow

The overriding dataflow of the Server Configuration File is seen below:

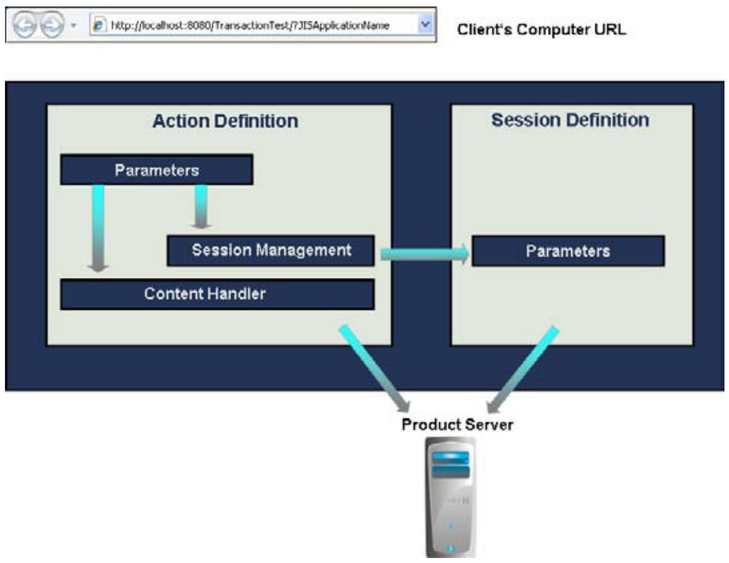


Figure 75. Server configuration file dataflow

The Server Configuration File contains two main elements:

- **`<SessionDefinitions>`**
- **`<ActionDefinitions>`**

<SessionDefinitions>

This element defines session definitions.

In the flowchart above, you can see how a session definition receives overriding parameters from the action definition.

<ActionDefinitions>

This element defines action definitions.

In the flowchart above, you can see how an action definition receives overriding parameters from the client and how the action definition overrides session parameters.

Session Definitions

A session definition contains the settings required for one host session. Session definitions are defined within the `<SessionDefinitions>` element.

A session definition specifies the settings for:

- Opening, maintaining and freeing host sessions and providing the host application with the information it requires for login.
- Initiating a JIS runtime Application and providing it with the information it requires.
- Receiving overriding data from the Transaction.

A session definition's settings may be overridden by the specific transaction it hosts.

Action Definitions

An action definition contains the settings required for one transaction. Action definitions are defined within the `<ActionDefinitions>` element.

An action definition specifies the settings for:

- Designating a session definition to house the transaction and providing this session definition with the information it requires for accommodating the transaction.
- Invoking an ACE method from within the webMethods JIS runtime and providing the method with the parameters it requires.
- Receiving overriding data from the Client.

A transaction's settings may be overridden by the client who requests the transaction.

Getting Started

The XML tags, which comprise the Server Configuration file, are, in fact, the building blocks of a whole new Markup Language, such as HTML. A firm understanding of XML is needed, in order to study the Server Configuration File in depth.

This section, however, provides several actions you can perform on the Server Configuration File, which is supplied with the webMethods JIS product, without the need to study the file in depth.

The following instructions are provided:

- Change a transaction's name.
- Specify a default JIS Application name.

Changing a Transaction's Name

When invoking a JIS Application from a browser, the Transaction's name is written right after the JIS Server's IP address and port number in the URL entry.

To change the XML Transaction Server's default transaction name:

1 Open the `ServerConfiguration.xml` file in a text editor and browse to the `<ActionDefinitions>` tag, where you will see the following line:

```
<ActionDefinition Name="TransactionTest" Default="Yes">
```

This is the tag that opens the default transaction named "TransactionTest".

2 Change the value of the `Name=` attribute to the name you desire.

3 Save the file.

To change the XHTML Client default transaction name:

1 Open the `ServerConfiguration.xml` file in a text editor and browse to the `<ActionDefinitions>` tag, where you will see the following line:

```
<ActionDefinition Name="XHTML">
```

This is the tag that opens the default transaction named "XHTML".

2 Change the value of the `Name=` attribute to the name you desire.

3 Save the file.

Specifying a Default JIS Application Name

When you enter a URL from a browser to invoke a JIS Application, you are required to supply a `JacadaApplicationName` parameter in the URL. You can avoid this by specifying a default JIS Application Name in the Server Configuration File. First, however, you must know the name of the transaction. The Transaction's name is written right after the JIS Server's IP address and port number in the URL entry.

To specify a default JIS Application name:

1 Open the `ServerConfiguration.xml` file in a text editor.

2 Under the `<ActionDefinitions>` tag, browse to the `<ActionDefinition>` tag, which has a `Name=` attribute bearing the correct transaction name.

Example: `<ActionDefinition Name="XHTML">`

This is the tag which opens the **XHTML** transaction's action definition.

3 Within this tag, continue browsing down to this line:

```
<Parameter Name="JacadaApplicationName" AssignTo="SessionManagement/
SessionProperties/Parameter
[@Name=JacadaApplication]">Test1</Parameter>
```

4 Change the tag’s value from `Test1` to the desired JIS Application name.

5 Save the file.

From now on, if you fail to supply the `JacadaApplicationName` parameter in the URL entry, the JIS Server invokes the JIS Application you specified in the tag’s value in the Server Configuration File.

Note: If you change this parameter’s `Name=` attribute, then in order to specify the JIS Application name in the URL request, you must enter the new parameter name instead of “JIS Application Name”.
For example: *`...ApplName=Test1`*

Writing the Server Configuration File

The Server Configuration File, `ServerConfiguration.xml`, must be placed in the `<InstallDir>/JacadaFiles/classes/` directory on the development PC.

You can either:

- Create a new file.
  -OR-
- Modify the existing `ServerConfiguration.xml` file provided with the product.

The Main Elements

Under the `DOCTYPE` line, create the main elements as follows:

```
<!DOCTYPE ServerConfiguration SYSTEM "ServerConfiguration.dtd">
<ServerConfiguration>
   <SessionDefinitions/>
   <ActionDefinitions/>
</ServerConfiguration>
```

The main elements in the Server Configuration File are:

`<Server Configuration>`	The root element.

`<SessionDefinitions>`	Defines session definitions. Each session definition is defined within one `<SessionDefinition>` element.
`<ActionDefinitions>`	Defines action definitions. Each action definition is defined within one `<ActionDefinition>` element.

The Server Configuration File's main elements are outlined below:

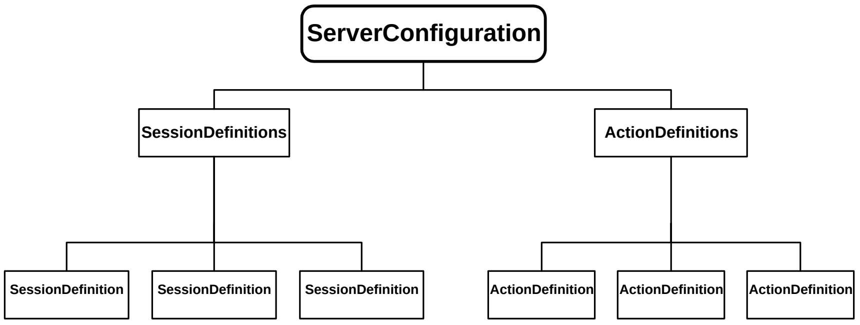


Figure 76. Server configuration file structure

Creating Session Definitions

`<SessionDefinition>` elements define session definitions. Each `<SessionDefinition>` element is identified by a `Name` attribute and may contain the following elements:

```
<SessionDefinition>
   <Parameters>
      <Parameter/>
      <Parameter/>
   </Parameters>
   <ClientLinkProperties>
      <BufferedResponse/>
      <SessionIdleTimeout/>
      <TrackSequenceNumbers/>
   </ClientLinkProperties>
   <JacadaApplication>
      <Name/>
   </JacadaApplication>
   <User>
      <Profile/>
      <UserId/>
      <Password/>
   </User>
   <SharedUserVariables>
      <Variable/>
      <Variable/>
   </SharedUserVariables>
</SessionDefinition>
```

The main elements in a session definition are:

`<JacadaApplication>` REQUIRED	Specifies the JIS Application that runs on the server. It is essential for session definitions to contain this element.
`<Parameters>`	Enables session settings to be overridden by a Transaction.
`<User>`	Specifies settings for logging onto the Host.
`<ClientLinkProperties>`	Defines the nature of the link to the Client.
`<SharedUserVariables>`	Defines variables used for session initialization.

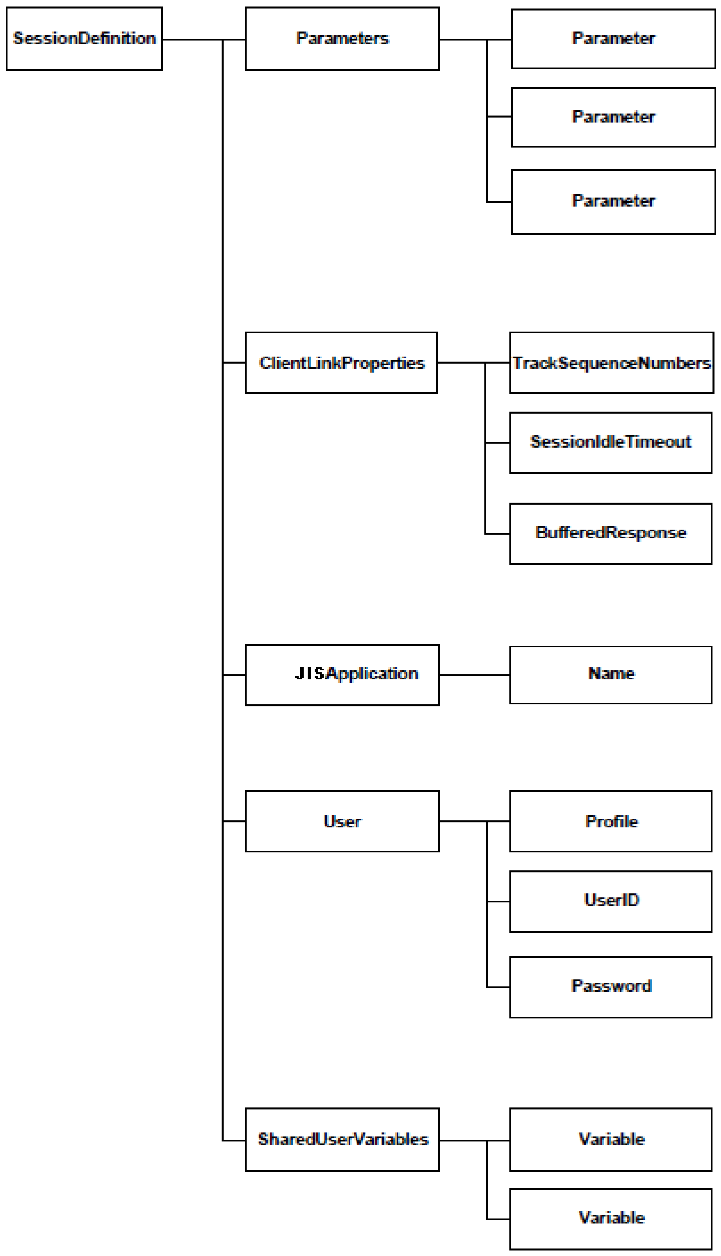


Figure 77. A typical <SessionDefinition> element

Creating Action Definitions

`<ActionDefinition>` elements define action definitions. Create one `<ActionDefinition>` element for each transaction. Each `<ActionDefinition>` element you create is distinguished by a different `Name` attribute and may contain the following elements:

```
<ActionDefinition>
   <Parameters>
      <Parameter/>
      <Parameter/>
      <Parameter/>
   </Parameters>
   <SessionManagement>
      <SessionProperties>
         <SessionDefinitionName/>
         <Parameter/>
         <Parameter/>
      </SessionProperties>
      <PostAction>
         <Free/>
      </PostAction>
      <SharedUserVariables/>
         <Variable/>
         <Variable/>
      </SharedUserVariables>
   </SessionManagement>
   <ContentHandler>
      <Transaction>
         <MethodName/>
         <MethodParameter/>
         <MethodParameter/>
      </Transaction>
   </ContentHandler>
</ActionDefinition>
```

The main elements in an action definition are:

`<Parameters>`	Enables action settings to be overridden by the Client.
`<SessionManagement>`	Specifies a session definition and overrides session settings.

`<ContentHandler>` or **`<DefaultHandler>`** REQUIRED	Define the transaction characteristics, i.e. specifying an ACE method or defining a default action. It is essential for an action definition to contain one of these elements.

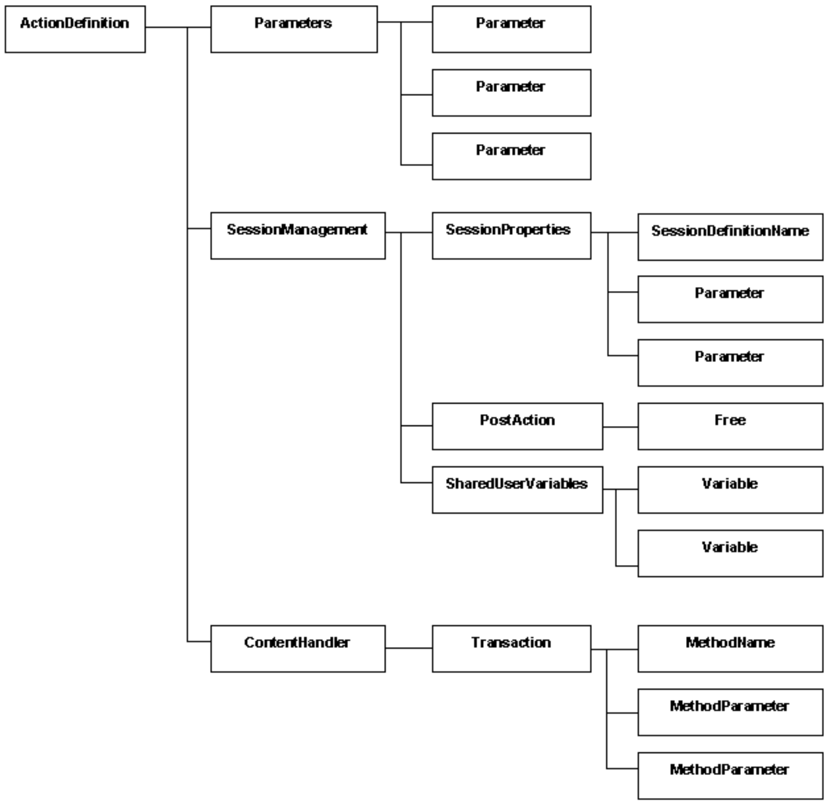


Figure 78. <ActionDefinition> with <ContentHandler>

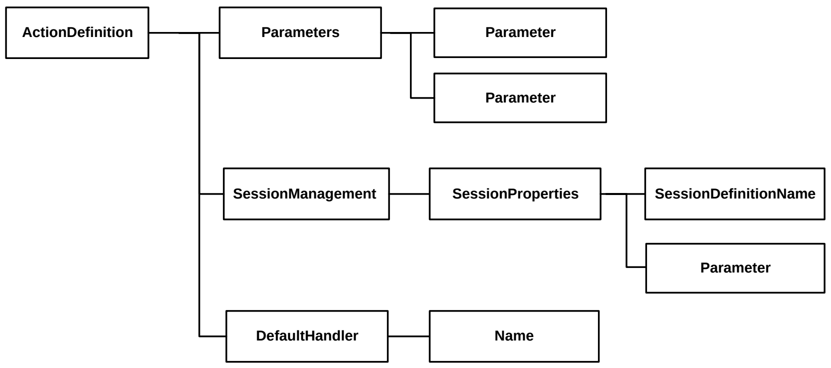


Figure 79. <ActionDefinition> with <DefaultHandler>

Server Configuration File Minimum Requirements

The simplest form of a working Server Configuration File is one which contains only the most essential elements.

You must create at least one session definition and one action definition as follows:

```
<ServerConfiguration>
   <SessionDefinitions>
      <SessionDefinition>
         <JacadaApplication>
            <Name>MyApplicationName</Name>
         </JacadaApplication>
      </SessionDefinition>
   </SessionDefinitions>
   <ActionDefinitions>
      <ActionDefinition>
         <ContentHandler>
            <Transaction>
               <MethodName>MyMethodName</MethodName>
            </Transaction>
         </ContentHandler>
      </ActionDefinition>
   </ActionDefinitions>
</ServerConfiguration>
```

Action Definition Essentials

The `<ActionDefinition>` element requires only the `<ContentHandler>` element or `<DefaultHandler>` (in case of a default action). The `<Transaction>` element contained in `<ContentHandler>` requires a `<MethodName>` element with an appropriate value, i.e. an ACE method's name. `<Transaction>` does not require `<MethodParameter>` elements.

Session Definition Essentials

The `<SessionDefinition>` element requires only the `<JacadaApplication>` element. The `<Name>` element contained in `<JacadaApplication>` must have an appropriate value, i.e. a JIS Application name.

Enabling Action Definitions to be Overridden by the Client

The following section explains how to enable default transaction settings to be overridden by the client.

- **On the client side**, you must specify the parameters that override the transaction settings.
- **On the transaction side**, you must “prepare” the action definition for receiving the client-specified parameter values and channel these values to override the appropriate transaction settings.

On the Client Side

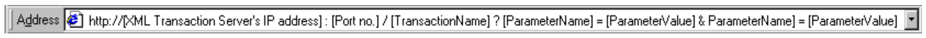
The Client’s relation with the transaction is defined in the URL command line. The client may either type a command line manually in the URL command entry box, or pre-define the URL command as part of an application.

Once the URL command line is executed, the request is sent to the XML Transaction Server, and this triggers the transaction. The transaction receives the parameter values specified in the client request and sends these values to override the appropriate action definition element values.

The URL command line includes the following:

- The XML Transaction Server’s IP address and port number.
- The transaction name.
- The parameter names to override and their values.

This is seen in Figure 80:



```
http://[XML Transaction Server's IP address] : [Port no.] / [TransactionName] ? [ParameterName] = [ParameterValue] & ParameterName] = [ParameterValue]
```

Figure 80. URL command line in an XML transaction

To manually set the Client URL to override transaction settings:

1 In any internet browser, select the URL entry box.
2 After the `http://` prefix, enter the XML Transaction Server’s URL address, a colon (:) character and the port number (usually 8080), followed by a slash (/) character.
3 Enter the transaction name, which is the value of the `Name` attribute of the desired transaction’s `<ActionDefinition>` tag, followed by a question mark (?) character.
4 Enter the parameter name, which is the **`Name`** attribute of the transaction’s `<Parameter>` tag you wish to override, followed by an equal sign (=) character.
5 Enter the desired value that will override this parameter.

6 If you wish to override an additional parameter, enter an ampersand (&) character and repeat steps 4-5. Do this for every additional parameter.

7 Press **Enter** to send this URL request to the XML Transaction Server.

Note: The above refers to manual activation of a transaction from a browser. The URL command line, however, may also be pre-defined as part of an Application.

On the Transaction Side

A transaction's relation with the client is defined in the action definition's `<Parameters>` element. This element contains several `<Parameter>` tags.

These `<Parameter>` tags are responsible for:

- Receiving tag values from the client.
- Passing on these values to action definition elements.

The transmitted values override the action definition elements' default values.

This is sketched below:

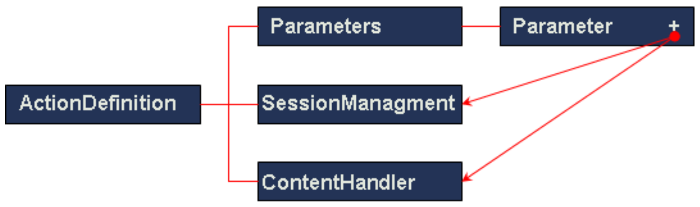


Figure 81. Values from URL request override default values

You can create as many `<Parameter>` tags as you need. Each of these `<Parameter>` tags may receive a tag value from the Client URL request, which overrides its value. Then, the `<Parameter>` tag sends its value to override an action definition element value.

Linking the Action Definition <Parameter> Tags

The `<Parameter>` tags must be linked to the appropriate elements.

An action definition's `<Parameter>` tag must be linked to:

Client	The HTTP request, in which a value is sent for this `<Parameter>` tag.

Transaction	The action definition element this `<Parameter>` tag's received value overrides.

To link an action definition <Parameter> tag to the Client:

1 Create a `<Parameter>` tag in the `<Parameters>` element. The tag should be empty.

2 Give the tag a `Name` attribute.

3 Give the `Name` attribute a value that is a logical name to which the client can relate.

To link an action definition `<Parameter>` tag to the action definition element it overrides:

1 Give the `<Parameter>` tag an `AssignTo` attribute.

2 Give the `AssignTo` attribute a value, which identifies the action definition element you wish the tag's value to override. The value of the `AssignTo` attribute is written in XPath syntax.

   For example, if the `AssignTo` attribute's value is "SessionManagement/SessionProperties/SessionDefinitionName", then the tag value overrides the `<SessionDefinitionName>` element tag located in the `<SessionProperties>` element located in the `<SessionManagement>` element of the current action definition.

Example 17. Client overrides Transaction

1. The Client sends a URL request for a transaction

In the following URL entry, the client establishes a connection with the XML Transaction Server residing in the IP address `190.200.74.84` (port no. 8080):

```
http://190.200.74.84:8080/NameList?ActionMethod=NameSearch&
MethodParameterA=John&MethodParameterB=Smith
```

The Client requests the following:

- Activate the "NameList" transaction, defined in the "NameList" action definition.
- Give the "ActionMethod" `<Parameter>` tag the value "NameSearch", and give the "ActionParameterA" and "ActionParameterB" `<Parameter>` tags the values "John" and "Smith".

2. The action definition <Parameter> tags are overridden by the Client

The “NameList” action definition `<Parameters>` element seen below contains three `<Parameter>` tags:

```
<Parameters>
   <Parameter Name="ActionName"
   AssignTo="ContentHandler/Transaction/MethodName"/>
   <Parameter Name="ActionParameterA"
   AssignTo="ContentHandler/Transaction/MethodParameter
   [@Name=MethodParameterA]"/>
   <Parameter Name="ActionParameterB"
   AssignTo="ContentHandler/Transaction/MethodParameter
   [@Name=MethodParameterB]"/>
</Parameters>
```

These `<Parameter>` tags’ values are overridden by the client URL request:

- The “ActionName” `<Parameter>` tag receives the value “NameSearch”.
- The “ActionParameterA” `<Parameter>` tag receives the value “John”.
- The “ActionParameterB” `<Parameter>` tag receives the value “Smith”.

3. The <Parameter> tags override the Transaction settings

The “NameList” action definition’s `<ContentHandler>` element is as follows:

```
<ContentHandler>
   <Transaction>
      <MethodName>purchase</MethodName>
      <MethodParameter Name="MethodParameterA">100</MethodParameter>
      <MethodParameter Name="MethodParameterB">200</MethodParameter>
   </Transaction>
</ContentHandler>
```

It contains the `<Transaction>` element, whose child elements are overridden:

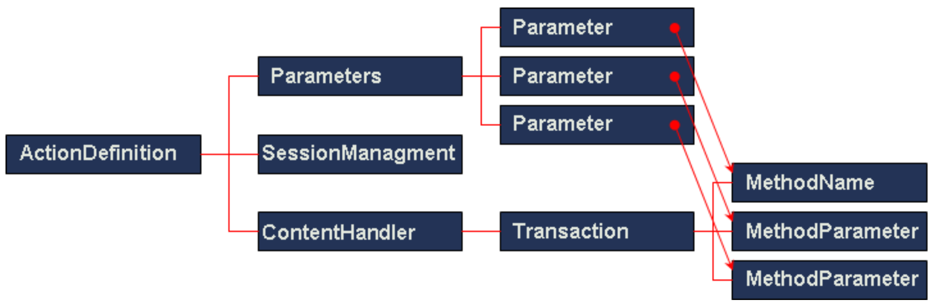


Figure 82. Children of transaction element are overridden by parameters

The "ActionName" <Parameter> tag

This tag’s `AssignTo` attribute is “ContentHandler/Transaction/MethodName”.

This means that its newly received value overrides the value “purchase” of the `<MethodName>` tag, located in the action definition’s `<Transaction>` element:

The "ActionParameterA" and "ActionParameterB" <Parameter> tags

These `<Parameter>` tags’ `AssignTo` attributes are set as:

- `ContentHandler/Transaction/MethodParameter [@Name=MethodParameterA]`
- `ContentHandler/Transaction/MethodParameter [@Name=MethodParameterB]`

This means that their newly received values override the values “100” and “200” of the `<MethodParameter>` tags named “MethodParameterA” and “Method ParameterB”, located in the action definition’s `<Transaction>` element.

To sum up the overriding process in this example:

1. The `<Parameter>` tag named “ActionName” receives the value “NameSearch” from the Client URL request, and this value overrides the action definition `<MethodName>` tag value “purchase”.
2. The `<Parameter>` tags named “ActionParameterA” and “ActionParameterB” receive the values “John” and “Smith” from the Client, and these values override the action definition `<MethodParameter>` tag values “100” and “200”.

Enabling Session Definitions to be Overridden by Actions

The following section explains how to enable a transaction to override the default settings of the session that houses it.

- **On the transaction side**, you must define the transaction-specific parameters that override the session settings.
- **On the session side**, you must "prepare" the session definition for receiving the transaction-specific parameters and channel these parameters to override the appropriate session settings.

On the Transaction Side

Each transaction's relation with the session is defined in the action definition's `<SessionManagement>` element sketched below:

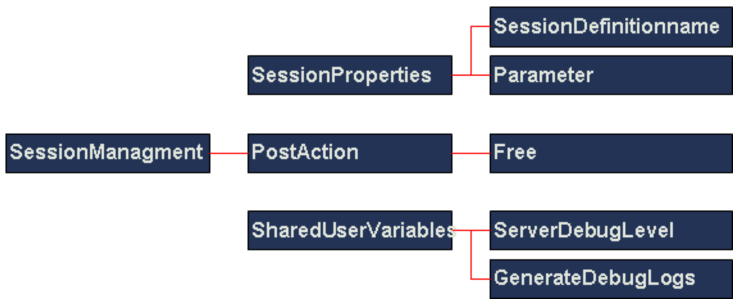


Figure 83. The action definition's SessionManagement element

To set the `<SessionManagement>` element:

1 Create the `<SessionProperties>` element.

2 In the `<SessionProperties>` element, create the `<SessionDefinitionName>` tag.

This tag's value is the name of the session this transaction will use.

3 In the `<SessionProperties>` element, create one or more `<Parameter>` tags, which define the session parameters you wish to override.

4 Give each `<Parameter>` tag a `Name` attribute. The `Name` attribute will be used by the session definition to identify the parameter.

5 Give each `<Parameter>` tag a value. This value will override the session settings.

Note: The next two steps are optional. Follow these steps if you wish to specify what happens to the session upon the transaction's completion.

6 Create the `<PostAction>` element. This element defines what happens to the session once the transaction is finished.

7 In the `<PostAction>` element, create the `<Free>` element tag. This tag specifies whether or not to free the session once the transaction ends.

If you give the tag a "Yes" value, then the session is freed immediately upon the transaction's completion, regardless of the time specified in the session definition.

On the Session Side

A session's relation with the transaction is defined in the session definition's `<Parameters>` element. This element contains `<Parameter>` tags.

These `<Parameter>` tags are responsible for:

- Receiving tag values from the transaction.
- Passing on these values to session definition elements.

The transmitted values override session definition elements' default values. This is sketched below:

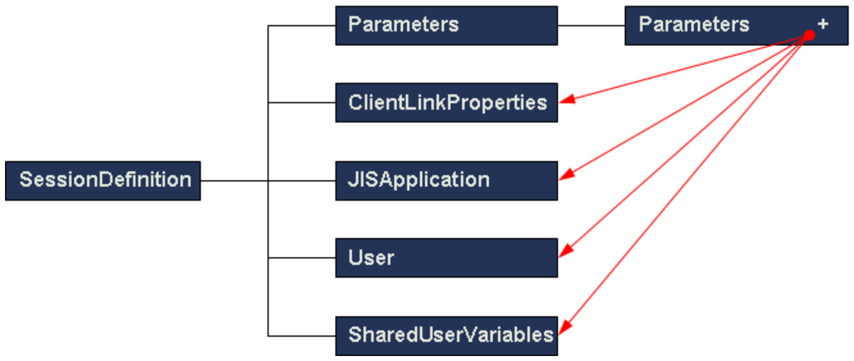


Figure 84. Parameter tags override session default settings

You can create as many `<Parameter>` tags as you need. Each of these `<Parameter>` tags corresponds to an action parameter, from which it receives a tag value, and then uses this received value to override one of the session definition's default settings.

Linking the Session Definition <Parameter> Tags

The `<Parameter>` tags must be linked to the appropriate elements. A session definition's `<Parameter>` tag must be linked to:

Transaction	The action parameter whose value overrides the `<Parameter>` tag.
Session	The session definition element this `<Parameter>` tag's received value overrides.

To link a session definition <Parameter> tag to the action parameter:

1 Create a `<Parameter>` tag in the `<Parameters>` element. The tag should be empty.

2 Give the tag a `Name` attribute.

3 Give the `Name` Attribute the same value as its corresponding action parameter. The corresponding action parameter is located in the action definition's `<SessionProperties>` element.

To link a session definition `<Parameter>` tag to the session definition element it overrides:

1 Give the tag an `AssignTo` attribute.

2 Give the `AssignTo` attribute a value, which identifies the session definition element you wish the tag's received value to override.

The value of the `AssignTo` attribute is written in XPath syntax.

For example: If the `AssignTo` attribute's value is "User/Profile", then the tag value will override the `<Profile>` element tag located in the `<User>` element of the current session definition.

Example 18. Transaction overrides Session

1. The action definition's <SessionManagement> element specifies a session and defines the overriding of session settings

An action definition's <SessionManagement> element is as follows:

```
<SessionManagement>
   <SessionProperties>
      <SessionDefinitionName>Session1</SessionDefinitionName>
      <Parameter Name="MyProfile">Manager</Parameter>
      <Parameter Name="MyApplication">Test</Parameter>
   </SessionProperties>
   <PostAction>
      <Free>Yes</Free>
   </PostAction>
   <SharedUserVariables>
      <VariableName="GenerateDebugLogs"/>
      <VariableName="ServerDebugLevel"/>
   </SharedUserVariable>
</SessionManagement>
```

It contains `<SessionProperties>`, `<PostAction>` and `<SharedUserVariables>` elements.

<SessionProperties>

Inside the `<SessionProperties>` element:

- The `<SessionDefinitionName>` tag's value is "Session1".
  This means that the transaction will use the session definition named "Session1".
- The `<Parameter>` tags are distinguished by their `Name` attributes:

"MyProfile"	Corresponds to the session parameter, whose `Name` attribute is also "MyProfile".
"MyApplication"	Corresponds to a session parameter, whose `Name` attribute is also "MyApplication".

These tags' values will override the values of the corresponding tags in the "Session1" session definition.

<PostAction>

The `<Free>` tag's value is "Yes". This means that the session is instructed to free itself upon the transaction's completion.

2. The "Session1" session definition receives overriding values

The session definition `<Parameters>` element contains two `<Parameter>` tags:

```
<Parameters>
   <Parameter Name="MyProfile"
   AssignTo="User/Profile"/>
   <Parameter Name="MyApplication"
   AssignTo="JacadaApplication/Name"/>
```

```
</Parameters>
```

These `<Parameter>` tags receive values from their corresponding `<Parameter>` tags, which reside in the action definition's `<SessionProperties>` element:

- The "MyProfile" `<Parameter>` tag receives the value "Manager".
- The "MyApplication" `<Parameter>` tag receives the value "Test".

3. The <Parameter> tags override the Session settings

The session definition's `<User>` and `<JacadaApplication>` elements are as follows:

```
<User>
   <Profile>Administrator</Profile>
   <UserId>JLENO</UserId>
   <Password>SATURDAY</Password>
</User>
<JacadaApplication>
   <Name Mandatory="Yes">MyAppl</Name>
</JacadaApplication>
```

Their child elements are overridden by the `<Parameter>` tags as follows:

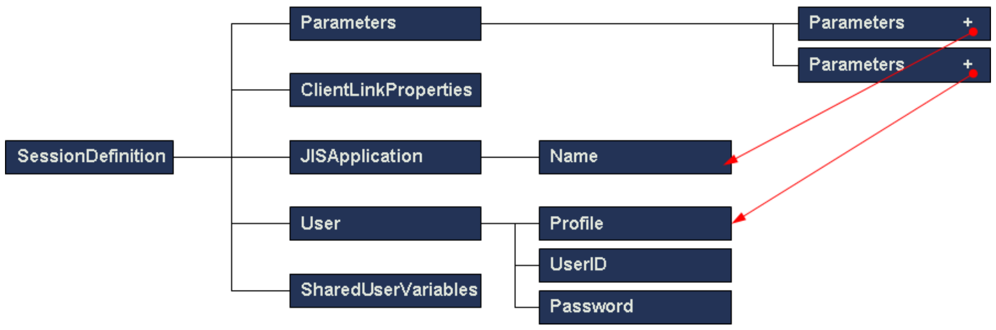


Figure 85. Parameters in session management element override values in session definition

The "MyProfile" <Parameter> tag

The `<Parameter>` tag's `AssignTo` attribute is set as "User/Profile". This means that its newly received value "Manager" overrides the value "Administrator" of the `<Profile>` tag, in the `<User>` element.

The "MyApplication" <Parameter> tag

The `<Parameter>` tag's `AssignTo` attribute is set as “JISApplication/Name”. This means that its newly received value “Test” overrides the value “MyAppl” of the `Name` tag, located in the session definition's `<JacadaApplication>` element.

To sum up the overriding process in this example

1 The action parameter value “Manager” will override the session parameter value “Administrator”.
2 The action parameter value “Test” will override the session parameter value “MyAppl”.

Example Server Configuration File

The following file is the Server Configuration file that is provided with the product. Instead of creating a new file, you may modify this one to suit your needs.

```
<!DOCTYPE ServerConfiguration SYSTEM "ServerConfiguration.dtd">
<ServerConfiguration>
   <SessionDefinitions>
      <SessionDefinition Name="Session1">
         <Parameters>
            <!-- A list of Session parameters, accessible to the Action. -->
            <Parameter Name="Profile" AssignTo="User/Profile"/>
            <Parameter Name="JacadaApplication" AssignTo="JacadaApplication/Name"/>
            <Parameter Name="Password" AssignTo="User/Password"/>
            <Parameter Name="UserId" AssignTo="User/UserId"/>
            <Parameter Name="UserLanguage" AssignTo="User/Language"/>
         </Parameters>
         <ClientLinkProperties>
            <BufferedResponse>Yes</BufferedResponse>
            <SessionIdleTimeout>180</SessionIdleTimeout>
            <TrackSequenceNumbers>No</TrackSequenceNumbers>
         </ClientLinkProperties>
         <JacadaApplication>
            <Name/>
         </JacadaApplication>
         <User>
            <Profile>John</Profile>
            <!-- Logon information. Used by Innovator. -->
            <UserId>UserName</UserId>
            <Password>UserPassword</Password>
            <Language></Language>
         </User>
         <!-- May be used by the UserInitApplication method. -->
         <SharedUserVariables>
            <Variable Name="GenerateDebugLogs">0</Variable>
            <Variable Name="ServerDebugLevel">0</Variable>
            <Variable Name="Var1">1</Variable>
            <Variable Name="Var2">2</Variable>
            <Variable Name="Var3">3</Variable>
         </SharedUserVariables>
      </SessionDefinition>
   </SessionDefinitions>
   <ActionDefinitions>
      <ActionDefinition Name="TransactionTest" Default="Yes">
         <Parameters>
            <!-- A list of Action parameters, accessible to the URI. -->
            <Parameter Name="SessionName" AssignTo="SessionManagement/
SessionProperties/SessionDefinitionName">Session1</Parameter>
            <Parameter Name="JacadaApplicationName" AssignTo="SessionManagement/
SessionProperties/Parameter[@Name=JacadaApplication]"/>
            <Parameter Name="ActionParameterA" AssignTo="ContentHandler/Transaction/
MethodParameter[@Name=MethodParameterA]">200</Parameter>
            <Parameter Name="ActionParameterB" AssignTo="ContentHandler/Transaction/
MethodParameter[@Name=MethodParameterB]">300</Parameter>
            <Parameter Name="MethodName" AssignTo="ContentHandler/Transaction/
MethodName">Method</Parameter>
            <Parameter Name="GenerateDebugLogs" AssignTo="SessionManagement/
SharedUserVariables/Variable[@Name=GenerateDebugLogs]"/>
            <Parameter Name="ServerDebugLevel" AssignTo="SessionManagement/
SharedUserVariables/Variable[@Name=ServerDebugLevel]"/>
            <Parameter Name="Var3" AssignTo="SessionManagement/SharedUserVariables/
Variable[@Name=Var3]">300</Parameter>
         </Parameters>
         <SessionManagement>
            <SessionProperties>
               <SessionDefinitionName>A Default Session</SessionDefinitionName>
               <!-- Action parameters sent to the Session. -->
               <Parameter Name="Profile">John</Parameter>
               <Parameter Name="JacadaApplication"/>
            </SessionProperties>
            <PostAction>
               <Free>No</Free>
            </PostAction>
            <SharedUserVariables>
               <Variable Name="GenerateDebugLogs"/>
               <Variable Name="ServerDebugLevel"/>
```

```
                <Variable Name="Var3">30</Variable>
                <Variable Name="Var4">40</Variable>
             </SharedUserVariables>
          </SessionManagement>
          <ContentHandler>
             <Transaction>
                <MethodName>A Default Method</MethodName>
                <!-- Action parameters sent to the Method. -->
                <MethodParameter Name="MethodParameterA">100</MethodParameter>
                <MethodParameter Name="MethodParameterB">200</MethodParameter>
             </Transaction>
          </ContentHandler>
       </ActionDefinition>
       <!--
         An example for the action that defines "subapplication" mode.
         It differs from it's "transaction" counterpart by the ContentHandler element.
         -->
       <ActionDefinition Name="SubapplTest">
          <Parameters>
             <Parameter Name="SessionName" AssignTo="SessionManagement/
SessionProperties/SessionDefinitionName">Session1</Parameter>
             <Parameter Name="JacadaApplicationName" AssignTo="SessionManagement/
SessionProperties/Parameter[@Name=JacadaApplication]">Test1</Parameter>
             <Parameter Name="GenerateDebugLogs" AssignTo="SessionManagement/
SharedUserVariables/Variable[@Name=GenerateDebugLogs]"/>
             <Parameter Name="ServerDebugLevel" AssignTo="SessionManagement/
SharedUserVariables/Variable[@Name=ServerDebugLevel]"/>
          </Parameters>
          <SessionManagement>
             <SessionProperties>
                <SessionDefinitionName>A Default Session</SessionDefinitionName>
                <Parameter Name="JacadaApplication">Test</Parameter>
             </SessionProperties>
             <SharedUserVariables>
                <Variable Name="GenerateDebugLogs"/>
                <Variable Name="ServerDebugLevel"/>
             </SharedUserVariables>
          </SessionManagement>
          <ContentHandler>
             <Subapplication/>
          </ContentHandler>
       </ActionDefinition>
       <!--Xhtml -->
       <ActionDefinition Name="Xhtml">
          <Parameters>
             <Parameter Name="SessionName" AssignTo="SessionManagement/
```

```
SessionProperties/SessionDefinitionName">Session1</Parameter>
            <Parameter Name="JacadaApplicationName" AssignTo="SessionManagement/
SessionProperties/Parameter[@Name=JacadaApplication]">Test1</Parameter>
            <Parameter Name="LauncherUserName" AssignTo="SessionManagement/
SessionProperties/Parameter[@Name=UserId]">UserName</Parameter>
            <Parameter Name="LauncherPassword" AssignTo="SessionManagement/
SessionProperties/Parameter[@Name=Password]">UserPassword</Parameter>
            <Parameter Name="Language" AssignTo="SessionManagement/
SessionProperties/Parameter[@Name=UserLanguage]"/>
            <Parameter Name="GenerateDebugLogs" AssignTo="SessionManagement/
SharedUserVariables/Variable[@Name=GenerateDebugLogs]"/>
            <Parameter Name="ServerDebugLevel" AssignTo="SessionManagement/
SharedUserVariables/Variable[@Name=ServerDebugLevel]"/>
         </Parameters>
         <SessionManagement>
            <SessionProperties>
               <SessionDefinitionName>A Default Session</SessionDefinitionName>
               <Parameter Name="JacadaApplication">Test</Parameter>
               <Parameter Name="UserId">John</Parameter>
               <Parameter Name="Password">John</Parameter>
               <Parameter Name="UserLanguage"/>
            </SessionProperties>
            <PreAction>
               <AddPostDataToParams>Yes</AddPostDataToParams>
               <AddHeaderDataToParams>No</AddHeaderDataToParams>
            </PreAction>
            <SharedUserVariables>
               <Variable Name="GenerateDebugLogs"/>
               <Variable Name="ServerDebugLevel"/>
            </SharedUserVariables>
         </SessionManagement>
         <ContentHandler>
            <Xhtml/>
         </ContentHandler>
      </ActionDefinition>

        <!-- CSS files for subapplication specific rules -->
        <ActionDefinition Name="XhtmlCSS" IncreaseSequenceNumber="false">
            <Parameters>
                <Parameter Name="JacadaApplicationName" AssignTo="SessionManagement/
SessionProperties/Parameter[@Name=JacadaApplication]"/>
                <Parameter Name="SessionId" AssignTo="SessionManagement/
SessionProperties/Parameter[@Name=SessionId]" Mandatory="Yes"/>
                <Parameter Name="LibraryName" AssignTo="ContentHandler/
MethodParameter[@Name=LibraryName]"/>
                <Parameter Name="SubapplName" AssignTo="ContentHandler/
MethodParameter[@Name=SubapplName]"/>
```

```
            </Parameters>
            <SessionManagement>
                     <SessionProperties>
                         <SessionDefinitionName/>
                         <Parameter Name="JacadaApplication"/>
                         <Parameter Name="SessionId"/>
                     </SessionProperties>
                 </SessionManagement>
            <ContentHandler>
                <XhtmlCSS>
                <MethodParameter Name="LibraryName"/>
                 <MethodParameter Name="SubapplName"/>
                </XhtmlCSS>
            </ContentHandler>
       </ActionDefinition>
        <!-- ********* The Special Actions ********* -->
      <!--
        Special action for session allocation.
        It hasn't a content handler, because
        its used for session initialization only. Any
        Subsequent requests on this allocated session
        will actually execute methods.
        -->
      <ActionDefinition Name="AllocateSession">
         <Parameters>
            <Parameter Name="SessionName" AssignTo="SessionManagement/
SessionProperties/SessionDefinitionName">Session1</Parameter>
            <Parameter Name="JacadaApplicationName" AssignTo="SessionManagement/
SessionProperties/Parameter[@Name=JacadaApplication]" Mandatory="Yes"/>
            <Parameter Name="GenerateDebugLogs" AssignTo="SessionManagement/
SharedUserVariables/Variable[@Name=GenerateDebugLogs]"/>
            <Parameter Name="ServerDebugLevel" AssignTo="SessionManagement/
SharedUserVariables/Variable[@Name=ServerDebugLevel]"/>
         </Parameters>
         <SessionManagement>
            <SessionProperties>
               <SessionDefinitionName>A Default Session</SessionDefinitionName>
               <Parameter Name="JacadaApplication"/>
            </SessionProperties>
            <SharedUserVariables>
               <Variable Name="GenerateDebugLogs"/>
               <Variable Name="ServerDebugLevel"/>
            </SharedUserVariables>
         </SessionManagement>
         <DefaultHandler>
            <Name>AllocateSession</Name>
            </DefaultHandler>
        </ActionDefinition>
        <!--
          Special action for session freeing.
          Again it hasn't a ContentHandler element.
          -->
        <ActionDefinition Name="FreeSession">
            <Parameters>
                <!-- The following decleration is required, although empty. -->
                <Parameter Name="SessionId" AssignTo="SessionManagement/
SessionProperties/Parameter[@Name=SessionId]" Mandatory="Yes"/>
            </Parameters>
            <SessionManagement>
                <SessionProperties>
                    <SessionDefinitionName/>
                    <Parameter Name="SessionId"/>
                </SessionProperties>
            </SessionManagement>
            <DefaultHandler>
                <Name>FreeSession</Name>
            </DefaultHandler>
        </ActionDefinition>
    </ActionDefinitions>
</ServerConfiguration>
```

XML Tag Reference

This appendix is an alphabetical reference to all the XML tags that are used in the Server Configuration File.

Each tag is provided with a brief explanation of its functions and attributes.

For each tag, the following information is provided:

- Parent element
- Required child elements
- Optional child elements
- Required attributes
- Optional Attributes
- Tag value

<ActionDefinition>

Parent Element	`<ActionDefinitions>`
Required Child Elements	`<ContentHandler>` or `<DefaultHandler>`
Optional Child Elements	`<Parameters>`, `<SessionManagement>`
Required Attributes	`Name=`
Optional Attributes	`Default=`

This element defines one transaction.

Name=*String*

Specifies the name of the transaction.

Default=*Boolean*

Specifies whether or not this transaction is the default one. If this attribute is set as “yes”, then the server addresses this transaction, unless specified otherwise in the URL command entry. If there is more than one default transaction, then the last one is addressed.

<ActionDefinitions>

Parent Element	`<ServerConfiguration>`
Required Child Elements	`<ActionDefinition>`
Optional Child Elements	none
Required Attributes	none
Optional Attributes	none

This element defines the different desired transactions. It must contain at least one `<ActionDefinition>` element, but you may probably want to define several `<ActionDefinition>` elements for different transactions.

<BufferedResponse>

Parent Element	`<ClientLinkProperties>`
Required Child Elements	none
Optional Child Elements	none
Required Attributes	none
Optional Attributes	none
Tag Value	Boolean

This element tag specifies whether or not the JIS Server waits until the transaction has finished before sending all the information in bulk back to the Client Application.

<ClientLinkProperties>

Parent Element	`<SessionDefinition>`
Required Child Elements	none
Optional Child Elements	`<BufferedResponse>`, `<SessionIdleTimeout>`, `<TrackSequenceNumbers>`
Required Attributes	none
Optional Attributes	none

This element defines the type and nature of the communication between the JIS Server and the XHTML client Application.

<ContentHandler>

Parent Element	`<ActionDefinition>`
Required Child Elements	`<Transaction>`
Optional Child Elements	none
Required Attributes	none
Optional Attributes	none

This element serves as a linking element between the current transaction and the JIS Server. Its child elements specify which ACE method the current transaction invokes.

<DefaultHandler>

Parent Element	`<ActionDefinition>`
Required Child Elements	`<Name>`
Optional Child Elements	none
Required Attributes	none
Optional Attributes	none

This element is used instead of the `<ContentHandler>` element to define a default action, which is a simple transaction that does not invoke an ACE method.

<Free>

Parent Element	`<PostAction>`
Required Child Elements	none
Optional Child Elements	none
Required Attributes	none
Optional Attributes	none
Tag Value	Boolean

This element tag specifies whether or not to free the session right after the completion of the transaction. If its value is `No`, the session remains open until the time specified under `<SessionIdleTimeout>` runs out.

<JacadaApplication>

Parent Element	`<SessionDefinition>`
Required Child Elements	`<Name>`
Optional Child Elements	none
Required Attributes	none
Optional Attributes	none

This element defines the JIS Application that runs on the JIS Server.

<MethodName>

Parent Element	`<Transaction>`
Required Child Elements	none
Optional Child Elements	none
Required Attributes	none
Optional Attributes	none
Tag Value	String

This element tag specifies the ACE method invoked by the current transaction. The tag's value is the name of the ACE method.

<MethodParameter>

Parent Element	`<Transaction>`
Required Child Elements	none
Optional Child Elements	none
Required Attributes	Name=
Optional Attributes	none
Tag Value	String

This element tag defines one ACE method parameter. The tag's value is the method parameter value.

<Name>

When under <DefaultHandler>

Parent Element	`<DefaultHandler>`
Required Child Elements	none
Optional Child Elements	none
Required Attributes	none
Optional Attributes	none
Tag Value	String

This element tag specifies the name of the default action. The tag's value is identical to the current `<ActionDefinition>` element's `Name` attribute.

<Name>

When under <JacadaApplication>

Parent Element	`<JacadaApplication>`
Required Child Elements	none
Optional Child Elements	none
Required Attributes	none
Optional Attributes	Mandatory=
Tag Value	String

This element tag defines the name of the JIS Application in which this transaction runs on the JIS Server.

Mandatory=*Boolean*

Specifies whether or not the JIS Server expects a value for this element. If the Mandatory attribute is set as `Yes`, failing to provide this element with a value results in a connection failure.

<Parameter>

When under <Parameters>

Parent Element	`<Parameters>`
Required Child Elements	none
Optional Child Elements	none
Required Attributes	Name= ,AssignTo=
Optional Attributes	Mandatory=

This element tag defines one overriding parameter.

There are two types of `<Parameter>` tags:

- **Action Parameters**, residing within an action definition, which receive their value from the Client and override local transaction settings.
- **Session Parameters**, residing within a session definition, which receive their value from the Transaction and override global session settings.

Name=*String*

Specifies the name of this parameter. This attribute is used for identification purposes. If this is a session parameter, then the Transaction overrides its value identifying the parameter by this name. If this is an action parameter, then the Client provides it with a value addressing the parameter by this name.

AssignTo=*String*

Identifies the element whose value this parameter's value overrides. This attribute's value is written in XPath syntax.

Mandatory=*Boolean*

Specifies whether or not the JIS Server expects a value for this element. If the Mandatory attribute is set as `Yes`, failing to provide this element with a value results in a connection failure.

<Parameter>

When under <SessionProperties>

Parent Element	`<SessionProperties>`
Required Child Elements	none
Optional Child Elements	none
Required Attributes	Name=
Optional Attributes	none

Contained in `<SessionProperties>`, this element tag defines an overriding parameter whose value overrides default session settings.

Name=*String*

Specifies the name of the parameter for identification purposes. The parameter must bear the same name as the session parameter it is to override.

<Parameters>

When under <ActionDefinition>

Parent Element	`<ActionDefinition>`
Required Child Elements	`<Parameter>`
Optional Child Elements	none
Required Attributes	none
Optional Attributes	none

This element serves as a linking element between the Client and the transaction. The `<Parameter>` tags it contains receive their value from the Client and override local transaction settings.

<Parameters>

When under <SessionDefinition>

Parent Element	`<SessionDefinition>`
Required Child Elements	`<Parameter>`
Optional Child Elements	none
Required Attributes	none
Optional Attributes	none

This element serves as a linking element between the Transaction and the Session. The `<Parameter>` tags it contains receive their value from the transaction and override global session settings.

<Password>

Parent Element	`<User>`
Required Child Elements	none
Optional Child Elements	none
Required Attributes	none
Optional Attributes	none
Tag Value	String

This element tag specifies the user's password. The tag's value is the password that is used to log onto the host.

<PostAction>

Parent Element	`<SessionManagement>`
Required Child Elements	none
Optional Child Elements	`<Free>`
Required Attributes	none
Optional Attributes	none

This element defines what will happen after the transaction has finished.

<Profile>

Parent Element	`<User>`
Required Child Elements	none
Optional Child Elements	none
Required Attributes	none
Optional Attributes	none
Tag Value	String

This element tag specifies the profile of the user on whose behalf the connection to the host is being made. The tag value is the profile that is used to log onto the host.

<ServerConfiguration>

Parent Element	none (root element)
Required Child Elements	`<ActionDefinitions>`, `<SessionDefinitions>`
Optional Child Elements	none
Required Attributes	none
Optional Attributes	none

This is the Server Configuration File’s root element. It has no attributes.

It contains two main elements:

- **`<ActionDefinitions>`**, for defining the transactions.
- **`<SessionDefinitions>`**, for defining the host sessions.

<SessionDefinition>

Parent Element	`<SessionDefinitions>`
Required Child Elements	`<JacadaApplication>`
Optional Child Elements	`<Parameters>`, `<SharedUserVariables>`, `<ClientLinkProperties>`, `<User>`
Required Attributes	Name=
Optional Attributes	none

This element defines one session definition, which contains the global settings of one host session.

Name=*String*

Specifies the name of the session definition.

<SessionDefinitionName>

Parent Element	`<SessionProperties>`
Required Child Elements	none
Optional Child Elements	none
Required Attributes	none
Optional Attributes	none
Tag Value	String

This element tag specifies the desired session. The value of this tag specifies which of the `<SessionDefinition>` elements is used to “house” this transaction.

<SessionDefinitions>

Parent Element	`<ServerConfiguration>`
Required Child Elements	`<SessionDefinition>`
Optional Child Elements	none
Required Attributes	none
Optional Attributes	none

This element defines the different possible host sessions. In this element, you must define at least one `<SessionDefinition>` element, but you may possibly want to define several `<SessionDefinition>` elements for different host session types.

<SessionIdleTimeout>

Parent Element	`<ClientLinkProperties>`
Required Child Elements	none
Optional Child Elements	none
Required Attributes	none
Optional Attributes	none
Tag Value	Integer (default is 180)

This element tag specifies how long, in seconds, a session remains open. If a session is not freed at the end of a transaction, it stays open until the specified time runs out. If a transaction gets “stuck”, then the session automatically frees itself when the specified time runs out. Default is 180 seconds (3 minutes).

<SessionManagement>

Parent Element	`<ActionDefinition>`
Required Child Elements	none
Optional Child Elements	`<SessionProperties>`, `<PostAction>`
Required Attributes	none
Optional Attributes	none

This element defines the relation between the transaction and the session.

<SessionProperties>

Parent Element	`<SessionManagement>`
Required Child Elements	none
Optional Child Elements	`<SessionDefinitionName>`, `<Parameter>`
Required Attributes	none
Optional Attributes	none

This element links the transaction to the session, by specifying a session definition and channeling specific transaction parameter values to override session settings.

<SharedUserVariables>

Parent Element	`<SessionDefinition>`
Required Child Elements	`<Variable>`
Optional Child Elements	none
Required Attributes	none
Optional Attributes	none

This element defines variables used for session initialization. This is an optional element, because these variables are not always needed. However, if used, it requires at least one `<Variable>` tag. Two user variables are included by default: `ServerDebugLevel` and `GenerateDebugLogs`. These variables enable the generation of logs for specific sessions.

<TrackSequenceNumbers>

Parent Element	`<ClientLinkProperties>`
Required Child Elements	none
Optional Child Elements	none
Required Attributes	none
Optional Attributes	none
Tag Value	Boolean

Track sequence numbers are automatically sent from the JIS Server to the Client Application, for synchronization purposes. However, not all Client Applications are inclined to receive these numbers. This element tag specifies whether or not the JIS Server should expect track sequence numbers from the Client Application.

<Transaction>

Parent Element	`<ContentHandler>`

Required Child Elements	`<MethodName>`
Optional Child Elements	`<MethodParameter>`
Required Attributes	none
Optional Attributes	none

This element defines the ACE method invoked by the current transaction.

<User>

Parent Element	`<SessionDefinition>`
Required Child Elements	none
Optional Child Elements	`<Password>`, `<Profile>`, `<UserId>`
Required Attributes	none
Optional Attributes	none

This element defines the user information required for logging on to the host.

<UserId>

Parent Element	`<User>`
Required Child Elements	none
Optional Child Elements	none
Required Attributes	none
Optional Attributes	none
Tag Value	String

This element tag specifies the name of the user on whose behalf this connection is being made. The tag's value is the name used for logging onto the host.

<Variable>

Parent Element	<SharedUserVariables>
Required Child Elements	none
Optional Child Elements	none
Required Attributes	Name=
Optional Attributes	none
Tag Value	String

This element tag defines one variable. The tag's value is the value of the variable.

Name=*String*

Specifies the name of the variable.

Chapter 12. Application Server Deployment

webMethods JIS applications can now run on a J2EE application server. This chapter discusses how to run webMethods JIS as a web application.

The following topics are discussed:

Web Application Deployment

The first step in deploying your application is to create a runtime installation. This is discussed in "Creating the Runtime Installation" on page 405.

After you have created the runtime installation, you must deploy it to an application server.

*.INI File Settings When Running as Web Application

When running as web application, the `jacadasv.ini` file contains only debug-related settings. Any other INI settings for your applications that you have until now placed in `jacadasv.ini` should be moved to the runtime INI file (<APPLNAME>.ini).

Creating the Runtime Installation

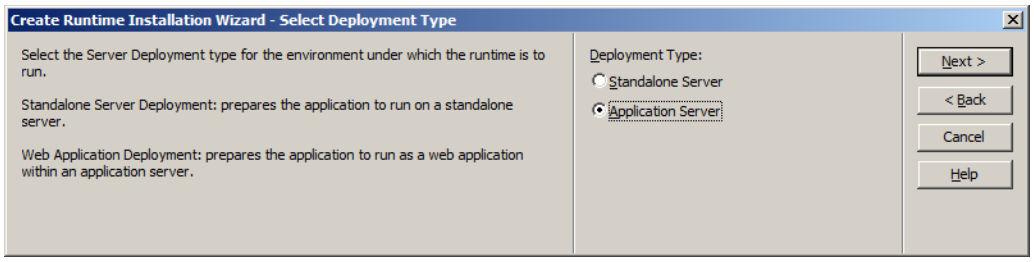
In order to deploy your application to an application server, you must first create a Runtime Installation. In the context of a web application, we also call the Runtime Installation a *deployment package*.The creation of a deployment package is required whether you are deploying to WebLogic, to WebSphere, or to Tomcat; to a Windows operating system or to Solaris.

Note: Before continuing, make sure that when you created the runtime you did so for use on the appropriate platform. This is done at Generate Runtime, in the wizard step JIS Server Platform. If you did this correctly, then, in the following directory
`<InstallDir>\JacadaFiles\classes\appls\<ApplName>\server\resources`
you should have a subdirectory named `bige`.

Create a runtime installation for web application deployment by using the Create Runtime Installation wizard on the Utility menu on the ACE menu bar and specifying Application Server deployment type.

Using the Create Runtime Installation Wizard

The runtime creation wizard is invoked by selecting Create Runtime Installation from the Utility menu on the ACE menu bar. On the dialog box for selecting deployment type, select Application Server Deployment.



Select the Server Deployment type for the environment under which the runtime is to run.

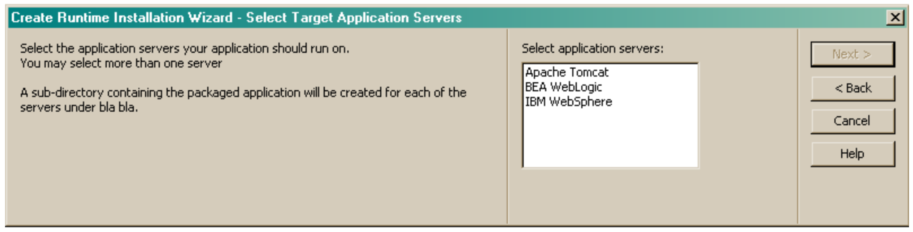
Standalone Server Deployment: prepares the application to run on a standalone server.

Web Application Deployment: prepares the application to run as a web application within an application server.

Deployment Type:
Standalone Server [✓ Application Server]

Figure 86. Installation: select Application Server deployment

The next step in the wizard asks you to indicate the desired target server or servers. The supported servers are listed. Select one or more of the servers and click Next.


Create Runtime Installation Wizard - Select Target Application Servers
Next >
< Back
Cancel
Help

Select the application servers your application should run on.
You may select more than one server

A sub-directory containing the packaged application will be created for each of the servers under bla bla.

Select application servers:
Apache Tomcat
BEA WebLogic
IBM WebSphere

Figure 87. Installation: select application server

Complete the Create Runtime Installation wizard as usual.

Details of the packaging process are saved in the `package.log` file. See "The package.log Files" on page 407.

The package.log Files

During the packaging process, a console window opens up to display progress messages. These progress messages are saved in a file named `package.log`, in the webMethods JIS install directory. In case an error is detected during the packaging process this `package.log` file will contain a reference to a more detailed log file.

The detailed log file is also called `package.log`. It records the operations performed during the packaging process, and is always created. The detailed `package.log` file is first written to the temp directory (as defined by the Windows environment variable %TEMP%). If the packaging process completes successfully the detailed log file is then copied to the
`<JIS Root>\JacadaFiles\deployment\<APPLNAME>`
directory. If the packaging fails, the detailed log file is left in the temp directory. In that case, a message notifying the user of the existence of the file in the temp directory is issued to the console window and also recorded in the less detailed `package.log` file in the webMethods JIS install directory.

Contents of the Application Server Runtime Installation

The results of the Create Runtime Installation wizard is a set of files and folders created in the directory
`<InstallDir>\JacadaFiles\deployment\<ApplName>`

The use of each of these folders and files is explained in detail later. A brief description of each follows.

- The `ForApplicationServer` folder contains an EAR file for deployment to an application server such as WebLogic or WebSphere.
- The `ForServletEngine` folder contains a WAR file for deployment to a web container such as Tomcat.

Deploying Your Application to WebSphere

Deploy your application to WebSphere:

1 Open the WebSphere administrative console.
2 Sign on to the administrative console.
3 In the navigation tree in the left panel of the administrative console, select **Application > Install New Application**.

4 Indicate the path of your application EAR file. It is named `<ApplName>.ear` and is located in the deployment package that you copied to the WebSphere machine, under the subdirectory `ForApplicationServer`.

   i.e., the path is:

   `<deployment package>\ForApplicationServer\<ApplName>.ear`
   After specifying the path to the EAR file, click **Next**.

5 On the Generate Default Bindings screen, click Next.

6 On the AppDeployment Options screen, click Next.

7 On the Map Virtual Hosts screen, click Next.

8 On the Map Modules to Application Server screen, click Next.

9 On the Summary screen, click Finish.

10 On the next screen, click Save to Master Configuration.

11 On the Save screen, again click Save.

Running Your Application with WebSphere under Windows

To execute your JIS Application:

1 Make sure that the WebSphere application server is running**.**

2 Open a browser window and go to URL

   ```
   http://<ServerMachine addr>:<9080>/<ApplName>/
   <ApplName>-webapp.html
   ```

   or just `http://<ServerMachine addr>:<9080>/<ApplName>/`

Note: 9080 is WebSphere’s default application port. If you have changed this default, your URL must reflect this.

The JIS Administrator

The JIS Administrator gives you the ability to query and change the configuration of your runtime environment without modifying the runtime executable. Use of the JIS Administrator is discussed in detail in the XHTML client book, under the heading “The Runtime Configuration Interface” in “Chapter 5 - Optimizing the JIS Server”.

Here, we explain how to install the JIS Administrator Runtime Configuration interface under WebSphere on the Windows operating system.

Deploying the JIS Administrator:

1 Open the WebSphere administrative console.

2 In the navigation tree in the left panel of the administrative console, select **Application > Install New Application**.

   Indicate the path of the JIS Administrator EAR file. It is named `JacadaAdmin.ear` and is located in the JIS Common Files directory that you created above, when you ran `windows-install.exe.` The path is therefore `<JacadaCommonFiles>\JacadaAdminApplication\ JacadaAdmin.ear`

   After specifying the path to the EAR file, click **Next**.

3 On the Generate Default Bindings screen, click Next.

4 On the AppDeployment Options screen, click Next.

5 On the Map Virtual Hosts screen, click Next.

6 On the Map Modules to Application Server screen, click Next.

7 On the Summary screen, click Finish.

8 On the next screen, click Save to Master Configuration.

9 On the Save screen, again click Save.

To run the JIS Administrator, open a browser and go to the following URL:

```
http://<WebSphere IPAddr>:9080/JacadaAdmin/admin
```

Note: 9080 is WebSphere's default application port. If you have changed this default, your URL must reflect this.

jacadasv.ini File Is Optional When Running as Web Application

When running your application under an application server, the `jacadasv.ini` file is not required, but can be used to set certain parameters. When the web application is first initialized, an empty `jacadasv.ini` file is created in the common directory. This `jacadasv.ini` file is read in subsequent runs.

The following parameters can be defined in the `GeneralParameters` section of the `jacadasv.ini` file when running as a web application:

- `RtDebugLevel`
- `RtDebugFilters`
- `RtDebugFileMaxSize`

The `MaxSessionInactivityTimeout` parameter, an *.ini file parameter when running with the Standalone Server, can be set as an initialization parameter of the web application when running under an application server. This is done using the `web.xml` configuration file.

See "JIS Server INI File Settings" on page 105 for details of how to use these settings.

When running as a web application, if you do not set the runtime root directory `(RtRootDir)` and runtime log files directory `(RtLogsDir)` in the `jacadasv.ini` file, the values are set internally to point to the JIS common directory.

JIS Application Debug Logging

If you want to generate a debug log for your application:

- Set the detail level of the log via the `RtDebugLevel` parameter in the `GeneralParameters` section of the `jacadasv.ini` file. The debug level can be set to any integer from 1 to 1000. The greater the integer, the greater the amount of information recorded in the log file. A debug level 70 produces an extremely detailed and voluminous log file. The default value is 1.

  Example:

  ```
  [GeneralParameters]
  RtDebugLevel=70
  ```

  By default, webMethods JIS creates a `logs` directory under the common directory that you specified in step 3 of the packaging procedure. The application logs are written to this directory. If you want the application log files to be created in a different directory, use the `RtLogsDir` parameter in the `GeneralParameters` section of the `jacadasv.ini` file.

  Example:

  ```
  [GeneralParameters]
  RtLogsDir=c:\mylogs
  ```

The name of the application log file is:

```
debug_jacada_<yyyy-mm-dd>_<random number>.log
```

The random number is used to distinguish between different servlet processes.

webMethods JIS can also produce session dumps. See "Enabling Dump File Generation" on page 169".

Chapter 13. Special to the XHTML Client

The webMethods JIS is available with more than one client type. For example, two of the most popular clients are the XHTML client and the Java client. This chapter discusses miscellaneous features and settings specific to the webMethods JIS XHTML client.

Modifying the Appearance of RMB Pop-up Menus

In the XHTML client, you can control the appearance of the right mouse button (RMB) popup menus through several Java APIs for the Window control.

Introduction

webMethods JIS lets you define pop-up menus that appear when a user clicks the right mouse button. You can define two kinds of such pop-up menus:

- A pop-up menu to appear when the main window is right-clicked. This pop-up menu lists the items defined for the Commands menu, if any.
- A pop-up menu to appear when a table is clicked. This pop-up menu lists the items defined for the List menu, if any.

The contents of these right mouse button pop-up menus is defined in ACE. In Design view, from the menu bar choose **Design > Subapplication Menu Editor**.

For a complete discussion of the use of the Subapplication Menu Editor, see the chapter about editing menus Design View in *webMethods JIS: Basic User's Guide*. If for a given window or table no Commands or List menu items are defined, no pop-up menu appears when the user right-clicks.

Related INI File Setting

This feature can be disabled with a setting in the `<Application_name>.ini` file, in the [Xhtml] section of the file:

```
RMBSupport=0
```

The default value of this setting is 1, which means the feature is working.

Controlling the Appearance of the Pop-Up Menus

You can control the appearance of the right-mouse button pop-up menus by means of several java methods.

APIs for Setting RMB Pop-Up Menu Characteristics

Set the menu’s foreground color

```
public void setRMBMenuForegroundColor (String color)
```

Set the menu’s background color

```
public void setRMBMenuBackgroundColor(String color)
```

Set the font of the menu items

```
public void setRMBMenuFontName(String fontName)
```

Set the font size of the menu items

```
public void setRMBMenuFontSize(int size)
```

Set the color used to highlight a specific item

```
public void setRMBMenuHighlightColor(String color)
```

Set the color of an item’s text when the item is highlighted

```
public void setRMBMenuHighLightTextColor(String color)
```

APIs for Querying RMB Pop-Up Menu Characteristics

Get the menu’s foreground color

```
public String getRMBMenuForegroundColor()
```

Get the menu’s background color

```
public String getRMBMenuBackgroundColor()
```

Get the color defined as the menu’s highlight color

```
public String getRMBMenuHighlightColor()
```

Get the name of the typeface used for the menu items

```
public String getRMBMenuFontName()
```

Get the font size of the menu items

```
public String getRMBMenuFontSize()
```

Get the text color used for a highlighted item

```
public String getRMBHighLightTextColor()
```

*.ini Settings for the XHTML Client

This section lists some of the *.ini settings that you should be especially aware of when using the XHTML client. Detailed instructions for using these settings can be found in "JIS Server INI File Settings", which begins on page 105.

[XHTML] Section of jacadasv.ini

All of the settings in the [XHTML] section of `jacadasv.ini` can alternately be defined in the runtime *.ini file (`<APPLNAME>.ini`). This is now the recommended procedure, as it will ease the task of migrating your application to a J2EE application server, should you decide to do so in the future. In a J2EE deployment, the settings of the [XHTML] *must* be located in the runtime *.ini file.

HTTPClient

When using the proprietary JIS Server for the XHTML client, the setting `HTTPClient` in the `[GeneralParameters]` section of the `jacadasv.ini` file must be set on.

PopupSupport ini setting

If you convert an application from the Java Client to the XHTML Client, be aware that you must set the `PopupSupport` INI setting in the [XHTML] section of the runtime INI file to 1 (its default is 0). This setting provides support for subapplication screens that are designated as pop-up windows when processing them in the New Subapplication wizard.

Keep Alive Implementation for the XHTML Client

In the XHTML client there is a mechanism for freeing unused sessions. A session is considered unused after a specific time period has elapsed with no activity.

(The *.ini settings discussed in this section are described fully inTable 15, "Jacadasv.ini: [XHTML] section", on page 119.)

The way that extended periods of session inactivity are handled is as follows:

- The client sends a "keep alive" HTTP request at an interval specified by the *.ini parameter `KeepAliveIntervalInSeconds`. The default value of this parameter is 60 seconds. Whenever the client receives a new XHTML page, the time counter is reset to zero. The `KeepAliveIntervalInSeconds` parameter can be set in the `jacadasv.ini` file or in the runtime *.ini file `(<applname>.ini).`
- Keep alive requests are sent by the client as long as the browser is open and within the application. When the JIS Server receives the keep alive request, it updates the last access time of the session. If the user closes the browser or exits the application, keep alive messages are no longer sent and the session is closed after the inactivity time-out `SessionIdleTimeout` is reached.
- The default inactivity time-out of XHML sessions is 3 minutes. This value is configurable via the `SessionIdleTimeout` in the `ServerConfiguration.xml` file for the JIS Server, and via the `session-timeout` parameter in the `web.xml` file when using a J2EE application server.
- The `MaxSessionInactivityTimeout` parameter limits the inactivity time of the session. For this time-out the keep alive messages are not considered as activity. This setting handles the situation of a browser left open for an extended period with no user activity. For the JIS Server, `MaxSessionInactivityTimeout` is configurable in the HTTP section of the `jacadasv.ini` file. For J2EE implementation, `MaxSessionInactivityTimeout` is set in the `web.xml` file. The implied units for this parameter are minutes, and the default setting is 60 (one hour).
- When a message box is open, no keep alive messages are sent, because browsers do not execute java scripts when a message box is open. Before the message box is sent, the client sends a special keep alive request. When the server receives this special request it stops waiting for keep alive requests and ignores the session inactivity parameter. (The session inactivity parameter for applications running on the JIS Server is `SessionIdleTimeout`; for applications running on a J2EE application server session activity is set via the `session-timeout` parameter in the `web.xml` file.) Only the `MaxSessionInactivityTimeout` is active when a message box is open. When the client returns from the message box the keep alive mechanism returns to its normal behavior.

Appendix A. Troubleshooting

This section describes some of the problems that you may encounter and some solutions that may help you when working with the XHTML client.

Problem / Message	Possible Reasons	Solution(s)
Unable to connect to the Server. (The message appears as an HTML on the browser).	JIS Server is not activated.	Activate the JIS Server
	Communication problems.	Contact System Administrator, and check if communication exists between the Client and Server. For example, use the Ping command followed by the server computer's address to verify that communication exists between the client and the server.
	You are behind a firewall.	In order to run the webMethods JIS delivered applications you will need to use HTTP communication, or enable communication through the Ports that webMethods JIS uses. The Ports are 1100 and 1101.
Application xxx is not installed on the Server. Try another Application.	Application was not found in the jacadasv.ini file.	Install the application on the Server computer
	The<applname>.ini file is missing.	Recompile the Application to generate the `<applname>.ini` file.

Problem / Message	Possible Reasons	Solution(s)
The maximum number of users are currently connected. Please try again later.	Maximum number of simultaneous clients was exceeded.	Try again later. The default port range for user TCP connections is 1024-5000. Change the `MaxUserPort` setting in the *Windows* registry to 65534 to increase the total number of available ports. **Warning:** Using the *Windows* registry editor incorrectly can cause serious, system-wide problems that may require you to reinstall *Windows*. Neither Software AG nor Microsoft guarantee that any problems resulting from the use of the registry editor can be solved. Use this tool carefully and at your own risk.
Generate runtime fails and you get the following message: *Error: Unable to delete the file: c://JISFiles/ classes/appls/<appname> ...cvrecord.dir*	Generate runtime cannot complete while the JIS Server is running.	Close the JIS Server.
When connecting to the JIS Server you get the following message: *Application not installed*	An application INI file <applname>.ini is missing.	Make sure all <applname>.ini files referenced in the jacadasv.ini actually exist.
The [ServerMachines] section is missing from jacadasv.ini	The section [ServerMachines] was not found in jacadasv.ini. jacadasv.ini is probably empty or corrupted.	Ensure jacadasv.ini is properly transferred to the runtime environment. On the mainframe, ensure it is in ASCII and not in EBCDIC.

Index