JIS Interface Server:

Java Client User's Guide

Version 9.3

September 2026
(originally released January 2005)

This document applies to JIS Interface Server Version 9.3 and to all subsequent releases.

Specifications contained herein are subject to change and these changes will be reported in subsequent release notes or new editions.

Copyright © 1992–2026 Software GmbH, Darmstadt, Germany and/or their suppliers. All rights reserved.

Detailed information on trademarks and patents owned by Software GmbH and/or its subsidiaries is located at http://documentation.softwareag.com/legal/.

Use of this software is subject to adherence to Software GmbH's licensing conditions and terms. These terms are part of the product documentation, located at http://documentation.softwareag.com/legal/ and/or in the root installation directory of the licensed product(s).

This software may include portions of third-party products. For third-party copyright notices and license terms, please refer to “License Texts, Copyright Notices and Disclaimers of Third-Party Products”. This document is part of the product documentation, located at http://documentation.softwareag.com/legal/ and/or in the root installation directory of the licensed product(s). Document ID: JIS-JAVACLIENT-UG-93-20260923

Table of Contents

About this Guide 17
Documentation Set 18
Document Conventions 19
Viewing the Documentation Online 20

Chapter 1. Making the Java Client Runtime Operational 21
Creating a Java Client Runtime 21
How The Compilation Process Works 22
Two Results of Generating a Runtime in JIS Interface Server 22
Setting Runtime Generation Options 22
How To Generate your Runtime 24
The Runtime Generation Process 25
Converting Images 25
Testing Your Runtime 26
Installing the Java Client Runtime 26
Creating the Runtime Installation 26
Installing Your Runtime 26
JIS Server Installation Options 27
Activating the Java Client Runtime 28
How Does the JIS Runtime Work? 29
The Java Client Runtime Deployment Architecture 29
The JIS Runtime on Windows 30
Creating a Runtime Installation 30
Installing Your Runtime on Windows 31
Activating the JIS Server on Windows 31
The JIS Runtime on Linux 32
Creating a Runtime Installation 32
Pre-Installation Checklist for the Linux Platform 33
Installing Your Runtime on Linux 34
Installing the Runtime Environment Using Samba 34
Mapping the PC to Linux: 34
Transferring the Runtime Environment to Linux 35
Installing Runtime Environment on Linux Using FTP 37
Installing Runtime Environment to a PC Directory 37
Compressing and Transferring Runtime Environment to Linux 39
Deploying Runtime Environment into Pre-defined Directory 39
Activating the JIS Server from Linux 40
The Jacadasv Script 41
Installing Multiple Applications on the Same JIS Server 41
The JIS Server Command Line Parameters 42
Operating the Runtime on a Client Computer 44

Chapter 2. The Java Client Runtime 47
HTML File Settings 47
HTML Parameters That Can be Changed by the User 48
GUI Settings 48
Localization 53
Communication 53
Debugging 55
HTML Pages 56
Session Initialization 57
Configurable User Messages 57
Scalability: Server Farm 58
Printing 59
Archive Files Provided by JIS 59
Server Side Archive Files 59
Client Side Archives Files 59
Java Client Administrator Related Archive 60
Signed and Unsigned Archive Files 61
The Application HTML Files Generated During Compilation 62
Running the Java Client from an Applet Using a Launcher 63
JacadaBasicLauncher 63
JacadaLoginLauncher 63
Running the Java Client from a Java Application 64
The Application’s Params File 64
Customized Application Settings 65
Running the Java Client Inside a Browser Window 66
Fitting the Application Window into the Browser 66
Changing the Application's Look 67
Limitations 67
Use Archives to Reduce Download Time of Java Classes 67
Referencing Application Specific Archive in Application HTML File 68
Referencing a JAR File in the Application HTML File 68
Improving Download Time of an Unsigned Application 68
Excluding the Downloading of the clhost.jar File 69
Optimizing Fonts for Non-Windows Platforms 69
The Font Substitution Resource File 69
The Resource File Location and Name 69
Contents of the Font Substitution Resource File 70
Debugging the Font Substitution Mechanism 71

Chapter 3. Optimizing the JIS Server 73
Other Factors Affecting Performance 73
JIS Server *.ini File Settings 74
Scalability 89
The Scalable System Structure of the JIS Server 89
Single Server-Computer System 90
Structure 90
Function 90
Multiple Server-Computer System 92

Structure 92
Function 93
Client Connection to the System. 93
Identifying Server Modules 94
The Integrator Process 95
Setting up the Scalable Server System 95
Customized jacadasv.ini File 96
General Structure of the jacadasv.ini File 96
The jacadasv.ini File is Composed of Sections. 97
Targeting ini Parameters to a Particular Machine or Node Level. 97
Precedence of Targeted ini File Sections 98
HTTP/S Communication 103
JIS Server Logging Support 104
JIS Server Logging Architecture 104
JIS Server Log Information Flow 105
The Server System Log Classes 106
SessionLog Log Class. 106
Viewing the SessionLog Output 108
Setting the LogClasses and Their jacadasv.ini File Parameters 108
LogClasses Section. 109
SessionLog Section. 109
XMLLog Section 109
XMLServer Section 109
How to Create a Server Log File 109
Advanced Logging Features 110
Controlling the Size of the Log File 110
The Start Log. 110
Debug Filters. 110
Analyzing Abnormal Runtime Termination 112
Information Included in Dump Files 112
Dump File Generation 113
Dump File Name and Location 113
Enabling Dump File Generation 113
Dump File Structure 114
Client Core Dump File 115
Session Core Dump File 118
Checking Server Configuration. 120
Server Configuration Checker 121
Reported Errors. 121
Reported Warnings 122
Enabling the Server Configuration Checker 123
Server Mode 123
Offline Mode. 123
Range of Valid Properties 124
JIS Administrator 127
Starting the JIS Administrator Command Line Utility 127
Starting JIS Server from the Server Machine 127
Connecting Online to the JIS Server 128

Debugging the JIS Administrator 129
The JIS Administrator Interfaces 130
The Server Monitor Interface 130
The Properties and Sessions Tabs 132
The License Tab 139
Operations you Perform Using the Server Monitor 140
The Runtime Configuration Interface 145
Running the JIS Server as a Windows Service 147
Registering the JIS Server in Windows 147
Parameters of JBSToService.exe 147
More Examples of the Use of JBSToService.exe: 149
Caution 149
Invoking the JIS Server as a Service 149
Logging off from the machine 150
Managing User Profiles 150
The User's INI Files Location 150
Creating a separate HTML file for each user 151
Using the LoginLauncher 151
Maintaining the User's Application INI File 152
To remove a user from an application or all applications 152

Chapter 4. Language Localization 153
How the Localization Feature Works 154
Localization Feature Workflow 154
Activating the Localization Feature 154
The Resource Files 155
The Original Resource File 155
The Translated Resource File 156
The General Resource File 157
Resource Maintenance 157
Setting the Runtime Localization Mechanism 157
Setting Through the Application HTML File 158
Setting Through the JacadaStarter API 158
String Types Handled by Localization 159
Debugging your Localized Application 160
How to Work in Debug Mode 160
On the Runtime Window 160
The Log File 161
ISO Language and Country Codes 162
Current Limitations 163

Chapter 5. Printing Features 165
Host-to-Client Printing 165
The Host-to-Client Printing Architectures 165
Host to Client Printer Emulation Connection via JIS Server 166
Making the Printer Emulation Operational 166
Configuring the Host to Recognize the Printer LU Name 167

Establishing Print Parameters in the <ApplName>.ini File 167
Adding HTML Parameter to Read Printer Emulation Archive 169
Initializing a Default Printer Emulation Session. 170
Tracing Printer Emulation Problems 170
Using the Printer 171
Print Options Dialog Box 171
Printing Via the Java Page Setup Dialog Box. 172
The Printer Emulation User Interface 174
Limitations 176
Troubleshooting 176
Extending the Printer Emulation 177
The Printer Emulation API Architecture 178
Creating a Printer Emulation Instance 178
Determining the Printer Emulation Type 179
The Default Settings 179
The Extension Class Path 180
Initializing an Extended Printer Emulation Session 180
The Application Runtime.ini File Settings 181
Special Runtime.ini Parameters Used with JIS Examples 183
HTML Parameters 183
Sending the Print Stream to the Client 184
Providing Client and Server Security Permissions 184
Client Security Permissions 184
Server Security Permissions 184
Examples of How to Use the Extended Printer Emulation 185
Saving Data on Server Example 185
HTML Printing Example 185
Printing via the Server and the Client 186
Printing the Client Window. 187
Activating the GUI Printing Feature 187
Eliminating the Print Setup Dialog 188
Modifying the GUI Printing Feature Through Code Extension 188
Granting Permission to Print the Client Window 189
Changing the Background Color of the Printed Window 190
Controlling the Scale of the Window's Printout 190
Printing the Client Host Screen. 191

Chapter 6. Extending the Java Code 193
The Client Java Code Produced During Compilation. 193
Java Sources in the Original Sub-directory 193
Java Sources in the User Sub-directory 194
Compiling the User's Java Sources. 195
Automatic Overwriting of User Files During Version Upgrading 195
Java .class Files 196
JIS's Javadoc Files 196
Working with the Java Code 196
About Event Handling 196
About Deprecated Methods 197

Where Can Code Extension Be Performed 198
Extending the Code of a Subapplication 198
Extending the Code of the Main Window 199
Extending the Code of All Subapplications 200
Understanding the Generated Java Code 200
Creating Controls 200
Creating Logic Peers 201
Event Handling. 201
Focus and Tabbing Management 203
Keyboard Management 203
Examples of Code Extension 204
Adding a Background Image 204
Adding Action Buttons to a Subapplication. 205
Adding Action Buttons to the Main Window Tool Bar 207
Querying a Button to Determine Its Characteristics 207
Adding Bubble Help to Components 208
Adding Animated Buttons to Subapplications 210
Updating Menu Items in Runtime 211
Defining Number and Length of Lines in Multi-line Edits 211
Selecting One Cell in Table Rows Using Right Click 212
Adding Content to a Table Cell 212
Handling Table Selection Events: Enabling the List Menu 213
Manipulating Host Originated Data 214
The Java Client RMB Floating Menus Support 216
Modifying the Default Floating Menus Behavior 217
Changing the Titles of Tab Control Folders During Runtime 219
Displaying Message Boxes 219
Creating Custom Validity Checks 221
Application-Wide GUI Settings 223
The Application's Color Scheme 223
Multi-Character Search in Combo Boxes 224
Data Sharing between Client and Server 225
Manipulating the Varpool from the Server Using Methods 225
Manipulating the Varpool from the Client Using Code Extensions. 226
The JacadaStarter's addWindow Method 227
Launching the Java Client from an Applet 228
The Java Client Launchers 229
Customizing a Launcher 230
Controlling the Java Client Application 232
Methods for Controlling the Java Client Application 232
Code Examples. 235
Implementing Localization Using the Java Localization API 239
Loading Specific Strings from Resource Files. 239
Initialization of Language Localization 239
Displaying System Messages in the Launcher Applet 240
Formatting Text 240
Technical Notes 241
ExternalFormat interface Functions 241

MyExternalFormat Class Example 242
Extending the Server's Java Code 243
When are Server Extensions Used 243
Server Java Sources Created During Compilation 243
Server Code Extension Types 245
Creating a Server Code Extension 245
The Server API 247
The Server API 247
JIS's javadoc Files. 249
Example of How to Write a Java Server Extension 249
Summary: Client vs. Server Code Extensions 251

Appendix A. Java Client Limitations 253
JIS Server Method Limitations 258

Appendix B. Troubleshooting 263

Appendix C. Directory and File Structures 267
Development Environment Directory Trees 267
The Users Runtime Directory Tree 270

Appendix D. Glossary of Terms 271

List of Figures

Runtime generation options dialog box 23
Two-tier architecture 27
Three-tier architecture 28
Series of events that takes place during runtime 29
JIS Server dialog box 32
Adding parameters in the HTML file. 48
Plugin warning dialog box 62
BasicLauncher 63
LoginLauncher 64
A Single Server 90
Control Flow When Starting a Session 91
A Server Farm 92
Identifying Server Modules 94
JIS Server logging architecture 104
Connect Online dialog box. 128
Server monitor interface. 130
Sessions tab 131
The Debug tab in the JIS Administrator. 136
The License tab in the JIS Administrator. 139
Stop JIS Server dialog box 142
Runtime Configuration interface 145
Language localization schema 153
Host-to-client printer emulation connection via JIS Server 166
MSIE Print dialog box running on Windows. 172
Java Page setup dialog box. 172
Printer emulation user interface 174
Printer Emulation Setup dialog box 175
Error message on printer fail 177
Optional ways to use the printer emulation 178
Method Parameters dialog box. 181
Print dialog box 188
Print dialog 192
Total label added to the bottom of the table’s second column. 216
Launching the Java client from an applet. 229
Extending the JacadaLoginLauncher 231
Method syntax error 260
Method syntax error 2 260
Java root directory file structure. 267
Client and Server Java source file directory structure 268
Client and Server Java class file directory structure 269
Users runtime directory structure 270

List of Tables

JIS Interface Server documentation set 18
Documentation conventions 19
Runtime generation options - Java tab settings 23
JIS Server command line parameters 43
GUI settings that can be changed by user 48
Localization settings that can be changed by user 53
Communication settings that can be changed by user 53
Debug settings that can be changed by user 55
HTML page settings that can be changed by user 56
Session Initialization settings that can be changed by user 57
Configurable User Message settings that can be changed by user 57
Scalability settings that can be changed by user 58
Printing settings that can be changed by user 59
Application HTML files generated during compilation 62
Server System's logging support components 104
Record parameters written to the SessionLog 106
Setting LogClasses and their jacadasv.ini parameters 108
Information included in dump files 112
SessionCoreDump parameters 114
HTML parameters 114
Client core dump file examples 115
Session core dump file examples 118
Reported errors 121
Reported warnings 122
Range of valid numeric properties in the jacadasv.ini 124
Parameters Used in the Connect Online Dialog Box 128
Properties in the Properties and Sessions tabs 132
Elements in the JIS Administrator Debug tab 137
Elements in the JIS Administrator License tab 139
Stop JIS Server parameters 142
Runtime Configuration interface parameters 145
Locale examples 162
Settings in the Printer Emulation Setup dialog box 176
Printer fail reasons and solutions 177
Default handler settings 179
Extension class paths and locations 180
Parameters in [TN3270 Printer] or [TN5250 Printer] 181
Special runtime.ini file parameters 183
HTML parameters 183
Choosing a scale for a printed window 191
Depreciated methods and their replacements 197
cst.server.export.api package 248
Client and server side extensions types and usage 251

Partially supported ACE controls 253
Partially supported features 256
Supported fonts 257
ACE versus Java font sizes 257
System-Triggered Methods included with ACE 261
Troubleshooting and solutions 263
Glossary of terms 271

List of Examples

Image file placement directory structure 25
Runtime installation file directory structure on Windows. 31
Creating a runtime installation on Linux 33
Locating the Shared Directory on Linux 35
Shared Directory examples on Linux 35
URL to the Linux machine 38
Copying applications 41
Add references to all applications in the jacadasv.ini 42
Copy HTML files 42
Adding parameters to the HTML file 48
Customized application settings 65
Optimize the application window size within the browser 67
Referencing a JAR File in the Application HTML File 68
Font substitution mechanism 69
File name examples 70
Font substitution 71
Debugging the font substitution mechanism 72
Example of jacadasv.ini for a multiple server-computer environment 98
Example of jacadasv.ini for a single server-computer system 98
Example of jacadasv.ini for a multiple server-computer system. 100
Server log file 110
Debug filters 111
Addfilter. 111
Original resource file 155
Translated resource file 156
Setting through the application HTML file 158
String types handled by localization 159
How to work in debug mode 160
Question Mark 161
Asterisk 161
Parameter containing the handler and classes 179
Activate the GUI printing feature through code extension 188
Preventing popup windows from being printed 189
Controlling the scale of the printed window 190
Activating Host screen printing via code extension 192
Adding controls in the createGUIControls method. 201
Adding a background image 205
Adding a Paste button 208
Displaying an animation sequence repeatedly. 210
Updating menu items during runtime 211
Adding contents to a table cell 213
Manipulating host originated data 215
Displaying a floating menu for a specific control type 217

Displaying a floating menu copied from the menu-bar 218
Display a floating menu not attached to a control 218
Adding the validity method 221
Writing the validity check 222
Using the validity check 223
Application's color scheme 224
Manipulating the varpool from the server using JIS Interface Server methods 225
Manipulating the varpool from the client using code extensions 226
Standard replacement of a Subapplication with a GUI 227
Preloading a class during initialization 228
Extending the JacadaLoginLauncher 231
SetCurrentPanel 233
activate 233
Skipping the login screen 235
Manipulating information, navigation and acceleration 237
Waiting 238
Initialization of language localization 239
Location of code for u_UserInitSubApplication 246
Extending GUTMs 247
JIS's javadoc file location 249
Placement of classes after runtime generation 269

About this Guide

JIS Interface Server is an automated development architecture that generates Java clients for enterprise applications. It uses JIS Interface Server's Automated Conversion Environment to convert character-based host screens into feature-rich graphical Java clients in 100% Java source code. Consequently, mainframe and iSeries applications can be delivered through an intranet, or over the Internet using a Java-capable client interface.

This book deals with the following topics:

- **Chapter 1 - "Making the Java Client Runtime Operational"**. This chapter walks you through the steps that bring your converted application to a fully operational JIS Interface Server runtime working on a Java-capable client.
- **Chapter 2 - "The Java Client Runtime"**. This chapter discusses advanced topics that are related to the Client side of the JIS Interface Server runtime. You will find here information about how to improve download time, how to enhance the runtime application by configuring various runtime settings, and the various possibilities for running the Java client.
- **Chapter 3 - "Optimizing the JIS Server"**. This chapter configure the JIS Server to serve large number of sessions and how to administer the JIS Server.
- **Chapter 4 - "Language Localization"**.This chapter provides you with all the information you require to implement the localization feature in your Application.
- **Chapter 5 - "Printing Features"**.This chapter introduces the Java client printing options:
  - The Printer Emulation feature enables you to command the host to send print jobs to a printer connected to your desktop computer, or to save them on the server.
  - The Client Window printing feature enables the printing of the window currently present on the client's screen.
  - The Client Host Screen printing feature enables the printing of the host screen currently displayed on the client's screen.
- **Chapter 6 - "Extending the Java Code"**- This chapter introduces you to a wide array of enhancements that you can perform on your application by extending the Java code.
- **Appendix A - "Java Client Limitations"**
- **Appendix B - "Troubleshooting"**
- **Appendix C - "Directory and File Structures"**
- **Appendix D - "Glossary of Terms"**

Documentation Set

JIS Interface Server 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. JIS Interface Server documentation set

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

JIS generates Java clients for host applications. In JIS 9.3, install JDK 21 for development and a Java 21 runtime on the server. Upgrade by packing the old application, unpacking it in a new JIS installation, regenerating the runtime, and recompiling extensions. Use Java Web Start / JNLP to launch current Java clients. The JIS 9.3 Release Notes provide the complete upgrade procedure.

This chapter describes:

- Creating a Java Client Runtime. The JIS Interface Server runtime is created on the conversion computer during the final stage of conversion, through a compilation process called Generate Runtime.
- Testing Your Runtime. After compiling your Application, check your runtime Application in the same type of environment you intend to deploy it.
- Making the Java Client Runtime Operational. To make the Java Client runtime operational, you must create a JIS Interface Server runtime, install it on the Server computer, and run it. This section provides you with background information about the JIS Interface Server runtime architecture, and detailed instructions on how to make the JIS Interface Server runtime operational for an Windows and Linux.
- Installing Multiple Applications on the Same JIS Server
- The JIS Server Command Line Parameters

Creating a Java Client 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 JIS Interface Server to compile your Application and create the Java classes and the JIS Server.

During the compilation process, Java code that corresponds to each Subapplication is generated. Each Subapplication has its own Java class. These are the Java classes that are sent by the Web Server when requested by the Client.

How The Compilation Process Works

Runtime generation produces Java sources and client launch files, then compiles the sources with the selected JDK 21. Generated javax.servlet references are transformed to jakarta.servlet before compilation. Recompile custom extensions after upgrading.

Two Results of Generating a Runtime in JIS Interface Server

The two results of generating a runtime in JIS Interface Server are:

- The Java client must be stored on the Web Server for the Clients to download and execute.
- JIS Server application specific classes that run on the Server Computer.

Setting Runtime Generation Options

Before you execute the Generate Runtime command, specify the runtime generation options.

To set the runtime generation options:

1. Open JIS Interface Server.
2. From the File menu, select Open Application, then select the Application that you want to compile and press OK.
3. From the Options menu, select Runtime Generation Options.

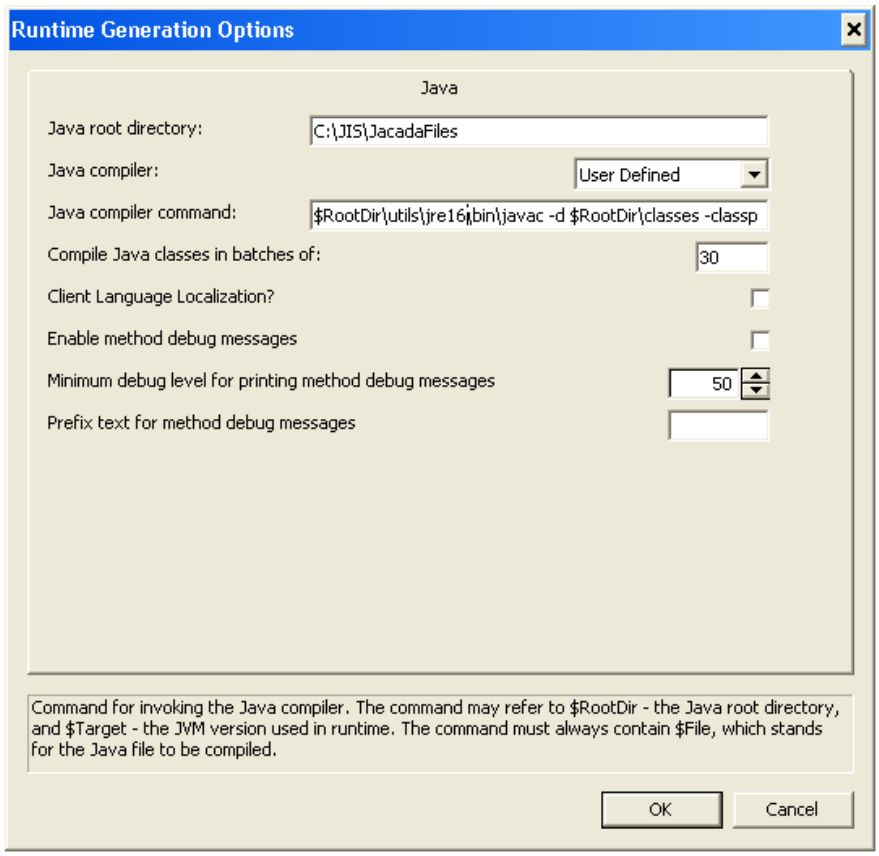


Figure 1. Runtime generation options dialog box

4 On the Java tab, specify the setting according to Table 3 on page 23.
5 When you have finished filling out the Java panel, press OK.

Table 3. Runtime generation options - Java tab settings (Sheet 1 of 2)

Java root directory	Choose the directory under which the Java Client files reside.
Java compiler	Choose a Java compiler.

Table 3. Runtime generation options - Java tab settings (Sheet 2 of 2)

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. Use the JDK 21 compiler selected during installation. Confirm the compiler command points to that JDK. After an upgrade, run jacc.bat for application and library user extensions, including server/user sources. 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. The compiler is automatically invoked as many times as necessary until all classes have been compiled. Reducing the number of classes compiled in each invocation 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 4 - "Language Localization" on page 153.

How To Generate your Runtime

The following is a description of how to generate a runtime 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 and the Java client.

2. In the Generate Runtime wizard, choose the following settings:
   - For a Runtime type, choose Java.
   - For the JIS Server Platform(s) option, choose one or more of the given platforms.
3. Continue the wizard to its final step.

   The runtime environment will be generated for the platform(s) you have specified in the Generate Runtime wizard.

The Runtime Generation Process

When generating a runtime, the Java client and Server and their associated sources are generated. The Java sources are automatically compiled. Compilation errors or warnings are displayed in the Generating the Runtime window. A more detailed log is written to the `makevgs.log` file in the directory under which JIS Interface Server is installed; for example, "`<InstallDir>\makevgs.log`".

Converting Images

To run the Java Client using a browser, all of the images that were used in the Application must be converted to an image format supported by Java. Images supplied with JIS Interface Server are automatically installed both in BMP and GIF format. Any other image must be converted and placed in the images directory under:

```
<InstallDir>\classes\appls\<ApplName>\images
```

Example 1. Image file placement directory structure

```
<InstallDir>\JacadaFiles\classes\appls\<ApplName>\images
```

When during the runtime generation process JIS Interface Server encounters images that do not exist in the images directory, a warning message is displayed in the Generating the Runtime output screen. Following this message you are requested to copy the missing image to the appropriate directory. The list of the missing images is written to the log file `makevgs.log` in the directory under which JIS Interface Server is installed.

> **Note:** Make sure that no two files have identical names (image.gif and image.jpeg, for example). If such is the case, then one of these files is randomly chosen.

Testing Your Runtime

In order to test your runtime, you must run the application. To run your application:

1. If the Application that you want to run is not already open within JIS Interface Server, then from JIS Interface Server's File menu select Open Application, then select the Application that you want to compile, and click OK.
2. From JIS Interface Server's File menu, choose Run Application.

Installing the Java Client Runtime

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

In order to make a Java Client runtime operational you must:

- Create a runtime installation
- Install the runtime
- Activate the JIS Server on the Server machine

In the following sections you will get background information about the JIS Interface Server runtime architecture, and detailed instructions on how to make the JIS Interface Server runtime operational running on Windows and Linux.

Creating the Runtime Installation

The runtime installation process involves copying the runtime environment from the computer JIS Interface Server 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

Both the Java client and the JIS Server application classes that were generated by Generate Runtime must be installed on the Server Computer. After creating a runtime installation, you are ready to install the runtime. The runtime can be installed on Windows and Linux, depending on the choice made earlier.

JIS Server Installation Options

There are two major installation options. In the first scenario, the host computer functions as a server. In this case, the JIS Server is installed directly on the host computer. This is referred to as the “two-tier” architecture model, since this configuration consists of two machines that communicate between each other.

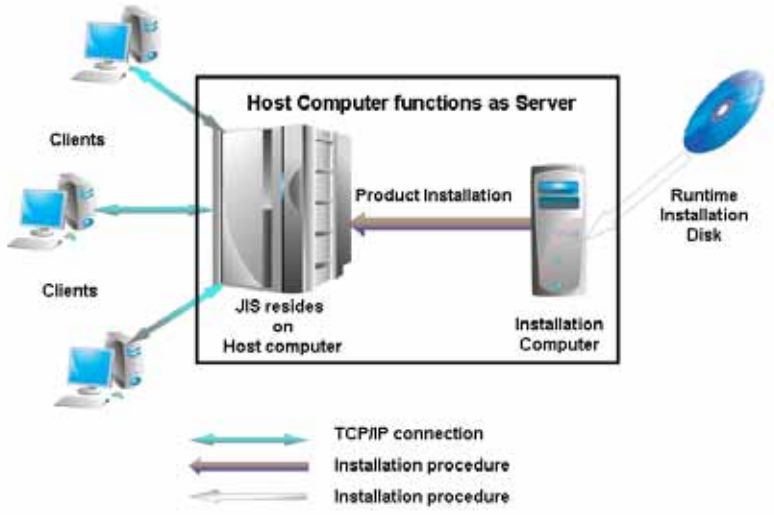


Figure 2. Two-tier architecture

In the second scenario, a server computer is designated, and the JIS Server is installed on the server computer. This is referred to as the “three-tier” architecture model, since this configuration consists of three machines that communicate between each other.

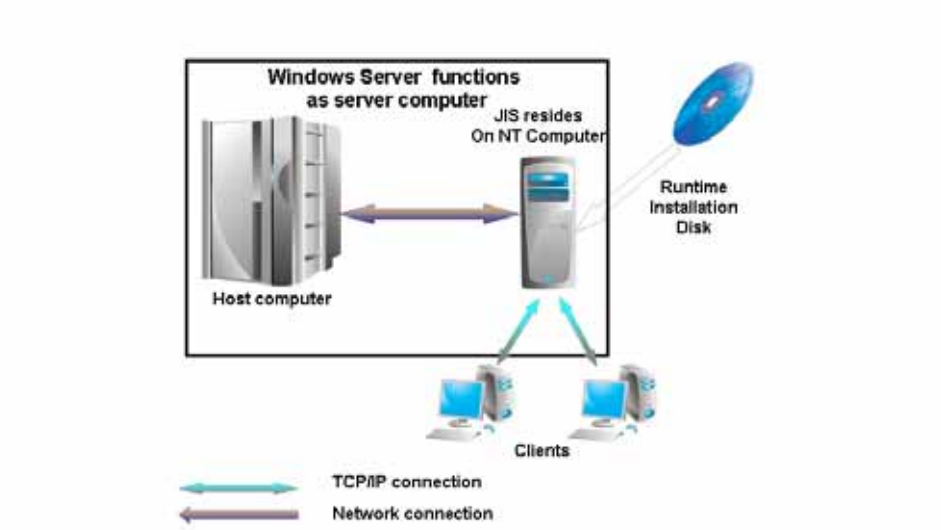


Figure 3. Three-tier architecture

Activating the Java Client Runtime

Once you have installed the JIS Server, you are now ready to operate the Java Client runtime. Operating the Java Client runtime involves activities on both the Server Computer and the Client Computer.

The following sections will give you a theoretical understanding of the way the JIS Interface Server runtime functions, and practical information about how to operate the Java Client from the Server computer and from the Client computer.

How Does the JIS Runtime Work?

Figure 4 illustrates the series of events that takes place upon connecting to the Server Computer. For simplicity, it is recommended to use the Jetty Web Server embedded inside the JIS Server.

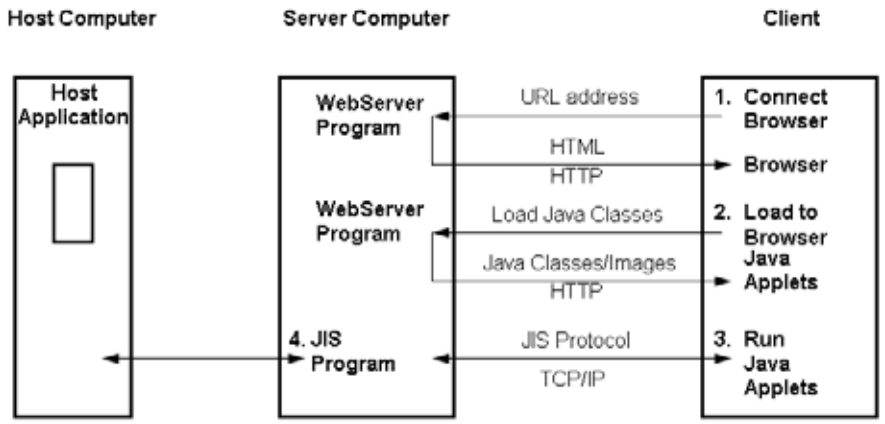


Figure 4. Series of events that takes place during runtime

The Java Client Runtime Deployment Architecture

The current JIS 9.3 launch sequence is:
1 Start the JIS server with Java 21 and open the generated client launch page or JNLP file.
2 Launch the Java Web Start client through the configured JNLP handler.
3 The Java client connects to the JIS server.
4 The server starts a 3270/5250 host session and sends the first window to the client. Browser applet launch is historical; see the Release Notes.

The JIS Runtime on Windows

In this section you learn how to install and operate the JIS Interface Server runtime on Windows.

Creating a Runtime Installation

To create a runtime installation:

1 Open the application in ACE and select Utility / Create Runtime Installation.
2 Select Stand Alone Server or Application server, Create IzPack Installation, Java runtime type, Full Runtime scope, and Windows platform. Set the installation directory and optional splash image.
3 Finish the wizard. The signed XML license is included in the runtime package. Copy the generated install folder to the target server and install Java 21 before running RuntimeInstallation.exe.

IzPack EXE RuntimeInstallation.exe installs the generated Windows runtime.

License The package includes SAGlicense.xml and its license.dat reference. Distribute the complete generated install folder. Use the signed XML license included in the runtime package.

The generated installation is placed under the application install folder:

```
<InstallDir>/appls/<ApplName>/install
```

Example 2. Runtime installation file directory structure on Windows

Distribute the generated application install folder and its RuntimeInstallation.exe package. The signed license is included.

Installing Your Runtime on Windows

Install Java 21 on the target server, then run RuntimeInstallation.exe from the copied install folder. Select the target installation directory and complete the wizard. Test the upgraded application and its Java extensions.

Launch RuntimeInstallation.exe from the generated application install folder.

```
<InstallDir>/appls/<ApplName>/install/RuntimeInstallation.exe
```

For the alternative Wise or manual packaging workflow, follow the generated setup.txt file. It lists the runtime files and their target locations. Include the signed XML license and its license.dat reference. The recommended JIS 9.3 procedure uses the Create IzPack Installation option.

Activating the JIS Server on Windows

On your PC, open the Start menu and select Programs > JIS > JIS Server.

When the JIS Server is activated, the screen in Figure 5 appears:

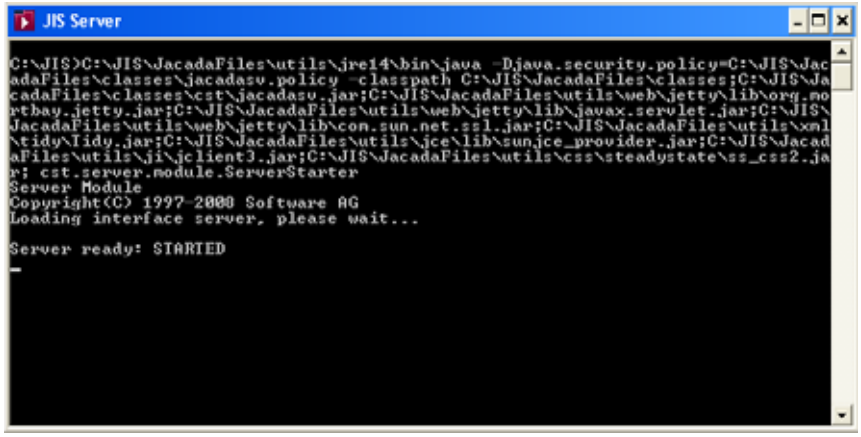


Figure 5. JIS Server dialog box

The JIS Runtime on Linux

In this section you learn how to install and operate the JIS Interface Server runtime on Linux. Before creating the runtime installation you need to have generated a runtime for the Linux platform. “Creating a Java Client Runtime” on page 21

Creating a Runtime Installation

To create a runtime installation:

1 In ACE, select Utility / Create Runtime Installation.
2 Choose Stand Alone Server or Application server, Create IzPack Installation, Java runtime type, Full Runtime scope, and Linux platform. Set the installation directory and optional splash image.
3 Finish the wizard. Copy the generated application install folder to the target Linux server. Install Java 21 and run java -jar RuntimeInstallation.jar. The signed XML license is included in the package.

The Linux IzPack output contains RuntimeInstallation.jar and the runtime files required by the selected package scope. The package includes SAGlicense.xml and its license.dat reference. Distribute the generated application install folder and install it with Java 21 using java -jar RuntimeInstallation.jar.

```
<InstallDir>/appls/<ApplName>/install
```

Example 3. Creating a runtime installation on Linux

<InstallDir>/appls/<ApplName>/install
Install Java 21 on the Linux server and run java -jar RuntimeInstallation.jar from this folder.

Pre-Installation Checklist for the Linux Platform

Before deploying the JIS Interface Server runtime on the Linux machine, check the following points.

- Test the regenerated server and Java client on Windows before Linux deployment.
- The runtime was generated for the Linux (x86) platform
- The runtime installation was created for the Linux (x86) platform.
- Check the supported Linux platform and install Java 21 on the server.
- Allow memory for Java 21, the generated runtime, and the expected session load.
- Verify that the Domain Name Server is properly configured.
- Verify that you have UNIX permission for transferring the runtime files and for executing the Java Server.
- See that you have a file sharing utility like Samba or NFS installed and configured on the Linux machine; alternatively, check that you can transfer files to the Linux machine via FTP. Samba and NFS are much easier to use than repeated FTPs. Samba is to be preferred over NFS, unless you already have NFS drivers installed on your PC. If you do transfer the files via FTP, verify that the files keep their names unchanged, without case translation.
- Check that JIS default ports 1100, 1101, and 2100 (or any alternative ports that you have configured the JIS Server to use) are not in use by another process on the Linux machine.

Installing Your Runtime on Linux

There are several types of utilities you can use to transfer directories and file structure from the PC to the Linux machine:

- Samba
- FTP
- Other utilities

In addition, you are provided with an installation wizard that automatically transfers the runtime environment to a directory you have pre-defined. The next section details the way you work with the JIS Server installation wizard. The following sections describe the ways to use the Samba and FTP utilities.

Installing the Runtime Environment Using Samba

If for whatever reason you are not able to use a file sharing utility to map a network drive to your Linux machine, skip ahead to the section “Installing Runtime Environment on Linux Using FTP” on page 37.

There are several utilities available for use on UNIX that let you view and manipulate its directories and file structure from the PC. Samba is a popular freeware filesharing utility that accomplishes this.

Use this utility to map a local drive on your PC to the Linux machine.

To install the runtime environment on the Linux machine

1. Map a local drive on your PC to the Linux machine. See next section for instructions.
2. Transfer the runtime environment to the Linux machine using the JIS Server Installation wizard. See “Transferring the Runtime Environment to Linux” on page 35.

Mapping the PC to Linux:

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

Use Network Neighborhood to map a PC drive to the Linux machine:

1. From the Network Neighborhood, find the name of the Linux machine and double-click on it.
2. Locate the “Shared directory” on Linux to which you wish to map the local drive.

Example 4. Locating the Shared Directory on Linux

```
\\<HostName>\home\JIS
```

Click on it with the right mouse button.

3 From the shortcut menu that appears, choose Map Network Drive.
4 In the Map Network Drive dialog box, choose a local drive.

Use Windows Explorer to map a PC drive to the Linux machine:

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—perhaps you want to use the “L” drive, for “Linux”.
3 Chose the appropriate path to the “Shared directory” which has been assigned to you on the Linux machine. If the desired path does not appear in the drop down list you can type it manually into the Path field.

Example 5. Shared Directory examples on Linux

```
\\12.34.56.78\JIS
```

or, with a DNS name

```
\\OURLINUX\JIS
```

4 Click OK.

Take note to record the Linux directory path. You will need to enter it in the JIS Server Installation wizard.

Transferring the Runtime Environment to Linux

Copy the install folder to Linux; use Java 21 with RuntimeInstallation.jar. The transfer wizard below is a legacy directory-mapping reference.

In the Installation wizard, in the Select Destination Directory dialog box, enter the directory under which the JIS Server files will be installed

- If you are using Samba, enter the runtime directory as defined when mapping your PC drive to the Linux machine.
- If you are using FTP, enter the name of a temporary directory.
- When working with Linux it is important to remember that UNIX is case sensitive. When creating an Application, JIS Interface Server 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.

In the Installing on Linux (x86) dialog box, provide the following:

The full path on the Linux to the directory where the application is installed

If the directory for your work on the Linux machine is:
`/export/home/JIS`
and you have designated the subdirectory `LINUXTST01` as the subdirectory to contains the runtime environment, then the full Linux directory for your runtime environment will be:
`/export/home/JIS/LINUXTST01`

Full path on the Linux machine to the Java utilities

The default is: `/usr/java`, the installation will automatically append /bin/java to this path.

URL for accessing the application

A URL including the Linux machine's address followed by the directory to which you are transferring the runtime environment, in the following format:
`http://<LinuxIPAddress>:8080/<SharedFolder>/<ApplDir>/`

For Example:

`http://ourlinux:8080/JIS/LINUXTST01/`

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.

Note: The client connects to the Application installed on the Linux machine via a web server. You must therefore have a web server installed on your Linux machine and you must map that web server to the directory in which the JIS Server is installed.

Installing Runtime Environment on Linux 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 using a file sharing utility, skip forward to “Activating the JIS Server from Linux” on page 40.

For JIS 9.3, transfer the generated application install folder to the Linux server. Install Java 21 there and run java -jar RuntimeInstallation.jar. Keep application names and paths unchanged during file transfer. The PC staging wizard below describes the earlier installation workflow.

To install the runtime environment on the Linux machine:

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

Installing Runtime Environment to a 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/Linux

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 Linux 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`, then enter: `<Drive>:\JIS\DEMO`.

In the Installing on Linux/x86 step, provide:

> **Note:** All directory path entries refer to the installation's destination on the Linux machine and not the temporary directory being used on the PC.

- Full directory path to destination directory as viewed on the Linux machine.

  Where you have designated the directory DEMO as the destination directory, then the full Linux directory for your runtime environment will be:

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

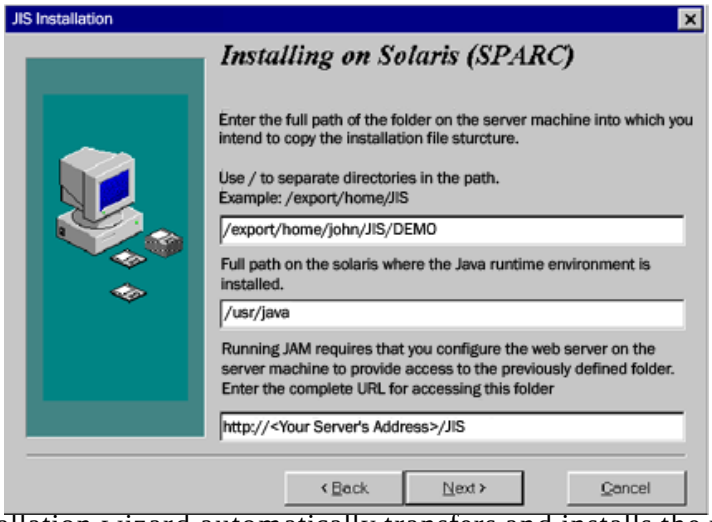
  For example: `/export/home/john/JIS/DEMO`
- Full path on Linux where the Java utilities are installed.
  The default is: `/usr/java`
- A URL including the Linux machine's address followed by the path to the destination directory.

  `http://Linux/~<UserName>/<DestinationDir>`

Example 6. URL to the Linux machine

`http://Linux/~John/DEMO`

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



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 Java client for your runtime Application.

Note: The client connects to the Application installed on the Linux machine via a web server. You must therefore have a web server installed on your Linux machine and you must map that web server to the directory in which the JIS Server is installed.

Compressing and Transferring Runtime Environment to Linux

To compress and transfer the runtime environment to the Linux 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 Linux machine.
3. FTP the compressed runtime environment to your target Linux machine

Deploying Runtime Environment into Pre-defined Directory

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

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

1. Using Telnet or any other means, open a session on the Linux 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 from Linux

Before you run the JIS Server on Linux, you should be aware that the file system on the Linux machine is case sensitive. Make sure that references to files are written in the exact way as the files themselves.

All *.ini files must be 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 Linux:

1. Open a session on the Linux 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 /<RTDir>/bin`

Note: `<RTDir>` must be replaced with the directory in which the Java installation resides.

3. Make a backup copy of file `jacadasv` by executing the command
   `cp jacadasv jacadasv_copy`
4. Execute the `dos2unix` command to change the end-of-line characters in file `jacadasv` from MS DOS-style to UNIX-style:
   `dos2unix jacadasv_copy jacadasv`
5. If this is the first time you run the JIS Server after having installed you must grant the user permission to execute certain JIS Server files.

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

   OR

   Type: `chmod 755 jacadasv linux/*.so`
6. Run the JIS Server. To do so, from the directory under which the runtime environment is installed type:
   `jacadasv`
7. Start up a web browser and type in the URL:
   `<LinuxIPAddress>:8080/<ApplName>.html`

Note: You can have the JIS Server started and run in the background during the Linux'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=/<RuntimeDir>/
set JAVA_INTERPRETER=/usr/java/bin/jre
set JAVA_CLASSES=/usr/java/lib/rt.jar
if ( $?LD_LIBRARY_PATH ) then
  setenv LD_LIBRARY_PATH $CST_DIR/bin/linux:$LD_LIBRARY_PATH
else
  setenv LD_LIBRARY_PATH $CST_DIR/bin/linux
endif
limit descriptors unlimited
exec $JAVA_INTERPRETER -classpath $JAVA_CLASSES\:$CST_DIR/classes:$CST_DIR/
classes/cst/jacadasv.zip:$CST_DIR/utils/xml/xml.jar
cst.server.module.ServerModule -d0 $*
```

Installing Multiple Applications on the Same JIS Server

The JIS Server has the capacity of running multiple Applications. This section explains how you install multiple Applications so they can run on the same server.

To install multiple Applications on your server

1. Compile all the Applications and install them on the JIS Server.
2. Choose one Application as the “main” Application. Its root directory will serve as a base for all other Applications.
3. Copy the other Applications from their respective `<RootDir>\classes\appls\<ApplName>` directories to the directory under which the “main” Application, say application ‘A’, was installed on the server.

Example 7. Copying applications

If application ‘A’ is installed on the server under `/home/runtime/A/classes/appls` and you are copying application ‘B’, then

copy: `/home/runtime/B/classes/appls/B`
to: `/home/runtime/A/classes/appls/B`

4 In the jacadasv.ini file, add references to all the applications you wish to run on the same server.

Example 8. Add references to all applications in the jacadasv.ini

```
[Applications]
B=
C=
[B]
WorkingDirectory= $RootDir/classes/appls/B/server/resources/
```

5 Copy the applications' HTML files to the `<RootDir>` directory of application 'A' (where the HTML file of application 'A' is stored on the server).

Example 9. Copy HTML files

Copy: `/home/runtime/B/B.html`
to: `/home/runtime/A`

The JIS Server Command Line Parameters

You can manipulate the way the JIS Server runs by adding command lines to its script file. Following is the list of parameters you can use for this purpose:

Table 4. JIS Server command line parameters (Sheet 1 of 2)

- d<Debug level>	Default: `-d1` **For 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 “50” produces an extremely detailed log file. A debug level of "1" is recommended when running the server in production environment. The debug facility is extremely useful for diagnosing problems that may occur during setup and testing of your JIS Server, but Software GmbH 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.
`-l<Debug log file directory>`	Default: `log to console` **For 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.
-m<number>	Specifies the maximum size of the server log file in bytes, if desired. Example: `-m1000000` Limit the JIS log file to 1 Mb in size.
-b<number>	Specifies the number of server log file revisions to keep.

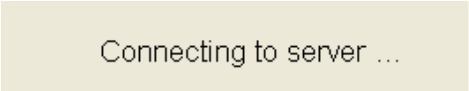
Table 4. JIS Server command line parameters (Sheet 2 of 2)

-i<path of server initialization file>	Default: `-i<User'sWorkingDir>\jacadasv.ini` **For Example:** `-ic:\JacadaFiles\jacadasv_example.ini`
-h	The JIS Server console displays the syntax of the 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 necessary when running the server as a background 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 110 for more information.

Operating the Runtime on a Client Computer

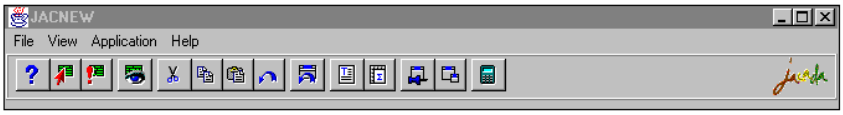
To run a Java Client application on a Client computer:

1 Open the generated JIS client launch page or JNLP file.
2 Start the Java Web Start client using the configured JNLP handler. The client loads the generated classes and connects to the JIS server. Use the release-specific JNLP and Java requirements in the JIS 9.3 Release Notes.



Note: Whenever a problem occurs in the connection between the Client Computer and the JIS Server, the Web Browser prompts a relevant HTML page. These pages contain detailed information about the possible reasons for the aborted connection.

3 Keep the Java Web Start client running. Once connected to the JIS server, the application main window appears and you can run the application. Closing the browser does not replace closing the client session.



Note: On the right hand side of the Main Window is the Java Client runtime icon. When this icon blinks, communication is executed between the Client and the Server.

4 After the main window appears, select Application > Run to begin running the application.

Chapter 2. The Java Client Runtime

JIS 9.3 Java clients launch through Java Web Start / JNLP or as Java applications. Browser applet support was retired in JIS 9.2.4. Applet-specific HTML and browser options in this chapter describe legacy applications; use the current Release Notes for launch and compatibility requirements.

In this chapter you will learn about:

- HTML File Settings
- Archive Files Provided by JIS
- Running the Java Client from an Applet Using a Launcher
- Running the Java Client from a Java Application
- Running the Java Client Inside a Browser Window
- Optimizing Fonts for Non-Windows Platforms

HTML File Settings

In order to run a Java applet, an HTML file is required. The HTML file is generated during the compilation process and stored under:

```
<InstallDir>\JacadaFiles\<ApplName>.html
```

Within the HTML file, certain parameters of the applet are pre-defined. These parameters can be changed by the user, in order to control certain runtime properties of the Java Client. The HTML file is not overwritten during subsequent executions of the Generate Runtime command, to prevent any editing performed on it from being destroyed.

In addition, you can add parameters to the html file.

Example 10. Adding parameters to the HTML file

```
<PARAM name = "UseHttp"   value = "TRUE">
```

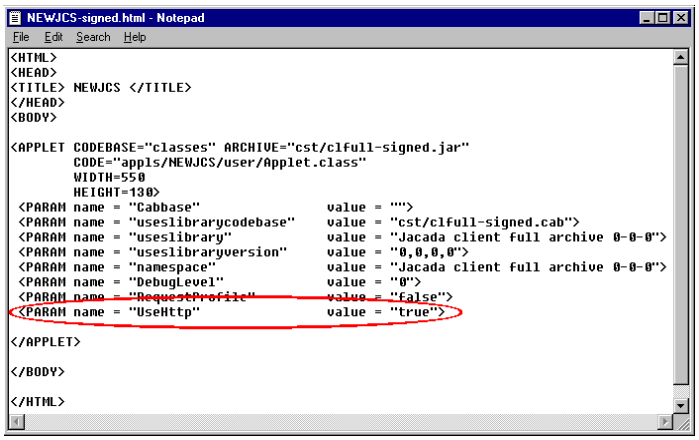


Figure 6. Adding parameters in the HTML file

HTML Parameters That Can be Changed by the User

The HTML parameters in Table 5 to Table 13 can be changed by the user.

GUI Settings

The GUI settings in Table 5 can be changed by the user.

Table 5. GUI settings that can be changed by user (Sheet 1 of 5)

Parameter	Description
HideMenus	When set to **TRUE**, the menu bar is not displayed in the Subapplication window. The default value is **FALSE**.
HideToolbar	When set to **TRUE**, the tool bar at the top of the main window and the Java Client runtime icon are not displayed. The default value is **FALSE**.

Table 5. GUI settings that can be changed by user (Sheet 2 of 5)

Parameter	Description
HideApplicationMenu	When set to **TRUE**, the **Application** menu is hidden. The default value is **FALSE**.
ComboFitWidth	Determines the width of an open combo box list: **FALSE** - the list opens as wide as the closed combo box. This is the default behavior. **TRUE** - the list opens as wide as the width of the longest item in it.
ComboFitHeight	When set to **TRUE**, combo boxes open as tall as needed to show all their items. The height is limited by the window's size. The default **FALSE** opens the drop-down combo as tall as defined in JIS Interface Server.
UseModalPopups	When set to **TRUE**, popup windows open as modal dialog boxes. This means that the user cannot give focus to the main window while the popup is open. The default value for this parameter is **FALSE**. See limitations under “Java Client Limitations” on page 253.
DefaultDateFormat	This parameter enables to set the format of date controls from the HTML. When the parameter does not appear, the locale's format is used. When the parameter’s value is "ShortLocale", the locale's format is used with a 2-digit year. When the parameter’s value is "LongLocale", the locale's format is used with a 4-digit year. Otherwise, you may determine your own format by typing it in as the parameter’s value, for example: "`dd/mm/yyyy`", "`yyyy-mm-dd`", etc.

Table 5. GUI settings that can be changed by user (Sheet 3 of 5)

Parameter	Description
HostScreenDisplaySize	The percentage that represents the ratio between initial host screen dimensions (height and width) and display resolution. The default is 70 percent. The actual window size is implementation dependent, since fonts are different for each Java version or Windows system. Note that the font size is determined by the screen's dimensions.
AnimatedLogoVisible	Determines whether or not to display the animated Software GmbH logo shown in the main window when the Server is running. The animated logo is displayed by default. Change the default value to **FALSE** to disable this feature.
AnimatedLogoImages	Contains the list of files where the images are placed. The file names must be separated by ',' or ';'. When the file names have no extension the default extension is ".gif".
AnimatedLogoLocation	Contains the directory where the logo images reside. This can be a path relative to the codebase, or a full URL address.
AnimatedLogoDelay	The time, in milliseconds, that each frame of an animated logo is displayed. The default is 250 milliseconds.
ButtonRolloverBackground	Defines the background color of a button when the mouse rolls over it. Used for giving the button a web-like look. The colors must be specified in the standard HTML format: RGB value as hexadecimal digits, with a # before the value, e.g. `value = "#666666"`.

Table 5. GUI settings that can be changed by user (Sheet 4 of 5)

Parameter	Description
ButtonRolloverForeground	Defines the foreground text color of a button when the mouse rolls over it. Used for giving the button a web-like look. The colors must be specified in the standard HTML format: RGB value as hexadecimal digits, with a # before the value, e.g. `value = "#FFFFFF"` for white.
ButtonCursorType	When the mouse rolls over a button, defines whether the shape of the cursor is that of a hand or an arrow. Used for giving the button a web-like look. The default value "Default" displays an arrow. Change the value to "Hand" to display a hand.
AllowSaveHostScreenImage	When set to **TRUE** the Emulator menu is added to the runtime menu bar. The Emulator menu contains the Save Host Screen Image menu item. Choosing this menu item prompts a dialog box in which the desired panel file name (.pnl) must be given. The default value is **FALSE**.
GUIPrintingBackground	Defines the background color of a printed window. The color must be specified in the standard HTML format: RGB value as hexadecimal digits, with a # before the value, e.g. `value = "# FFFFFF"` for white. This setting also affects the background color of group boxes, frames, radio groups, tab folders, labels, check boxes and radio buttons. See “Printing the Client Window” on page 187.
GUIPrintingScale	In percents, defines the scale of the window’s printout. For example: `value = "70"` causes the printout to be 70% of its full size. The default is 100 percent. Note: do not write the “%” character.
GUIPrintingInMonochrome	When set to **TRUE**, the GUI is printed in pure black and white. The default value is **FALSE**.

Table 5. GUI settings that can be changed by user (Sheet 5 of 5)

Parameter	Description
ApplicationIconLarge	Defines the location of the icon used in the **About** dialog box. This choice overrides the default icon. All Java supported image formats are valid. When no extension is given, .GIF is the default.
ApplicationIconSmall	Defines the location of the icon used for the **System** menu. This choice overrides the default icon. All Java supported image formats are valid. When no extension is given, .GIF is the default.
UseMultirowTabFolders	Defines whether the multi-row folder tabs feature is enabled or not. When set to **FALSE**, the folder tabs are lined up in one row regardless of whether they fit in the screen or not. The default values is **TRUE**.
RunInsideBrowser	When set to **TRUE**, the client runs inside the browser. When **FALSE** (the default value), the client runs in a separate window.
WindowHScrollIncrement	Defines the number of pixels a single click of the increment/decrement buttons of the horizontal scroll bar scrolls the Subapplication window. The default value is 10.
WindowVScrollIncrement	Defines the number of pixels a single click of the increment/decrement buttons of the vertical scroll bar scrolls the Subapplication window. The default value is 10.
WindowScrollRepeatInterval	Defines the time interval in milliseconds between each repeated firing, when clicking and holding the increment/decrement buttons on the horizontal/vertical scroll bars. The default value is 100 milliseconds.

Localization

The following localization settings can be changed by the user:

Table 6. Localization settings that can be changed by user

Setting	Description
Locale_Language	Defines the language that is used in the runtime Application. The value takes a two letter language code which is derived from the ISO 639 standard. For example: "fr" for French, "sp" for Spanish, etc. For more information, see Chapter 4 - "Language Localization" on page 153.
Locale_Country	Defines the language as spoken in a specific country that is used in the runtime Application. The value takes a two letter country code which is derived from the ISO 3166 standard. For example: "AU" for Australia, "CH" for Switzerland, etc. For more information see Chapter 4 - "Language Localization" on page 153.
Locale_Variant	Defines the language variant as spoken in a specific country that is used in the runtime Application. The value is vendor and browser-specific. For more information see Chapter 4 - "Language Localization" on page 153.

Communication

The following communication settings can be changed by the user:

Table 7. Communication settings that can be changed by user (Sheet 1 of 3)

Setting	Description
CommTimeOut	Specifies the time the Client waits for a response from the Server, before prompting the user with the message "Server not responding". The default value is 50 seconds.

Table 7. Communication settings that can be changed by user (Sheet 2 of 3)

Setting	Description
UsePorts	Defines whether or not to use the two direct communication ports for connecting the Client and the Server. The default value is **TRUE**.
ConnectPort	Determines which port is used in case the port file is not found. The default value 0 means that the default port 1100 is used.
ConnectPortURL	Indicates the full or partial path to the port file. This allows overriding the default place of the port file which is in the `classes` directory, or one directory level above that.
Server	Java Client's server name. You require the name when running the Java Client as an Application. The default name is null. In this case the Server is the one from which the Java classes are downloaded.
UseHttp	Defines whether to use an HTTP communication method. The default value is **FALSE**.
HttpAddr	Overrides the automatically calculated URL address. Allows for customized setting of the Web Server and for HTTPS communication even when classes are downloaded using HTTP. For Example: HttpAddr = http://localhost:80/servlet/JISProxyServlet Note that the port number is required even when it is default port 80.

Table 7. Communication settings that can be changed by user (Sheet 3 of 3)

Setting	Description
HttpReuseConnection	Determines whether clients working on the same JVM use a single connection to the servlet. When set to **FALSE**, the connection reuse mechanism is disabled. The default is **TRUE**.
ConnectionRetryTimeout	Determines the maximum total time in seconds during which the client continues its attempts to establish a connection with the server. The default is 20 seconds.
ConnectionRetryInterval	Determines the maximum time, in seconds, between connection attempts. The actual time is calculated randomly but does not exceed the specified value. The default is 10 seconds.

Debugging

The following debug settings can be changed by the user:

Table 8. Debug settings that can be changed by user (Sheet 1 of 2)

Setting	Description
DebugLevel	Controls the amount of debug information that can be logged to the Java Console. **For Example:** 50=detailed debug printing, 1000=full debug printing. The default **0** specifies no debug printing,
LocaleDebugMode	Determines whether the Application prints localization-related information to the debug log. The Application runs in localization debug mode when the parameter is set to **TRUE**. The default is **FALSE**.

Table 8. Debug settings that can be changed by user (Sheet 2 of 2)

Setting	Description
DebugTimeStamp	When **TRUE**, each line in the client debug log is preceded by a timestamp. The default is **FALSE**.

HTML Pages

The following HTML page settings can be changed by the user:

Table 9. HTML page settings that can be changed by user

Setting	Description
ExitPage	Defines which html to move to after the Application is finished. The default value is to remain in the current HTML page.
ConnectionFailedPage	Defines an html page that is displayed when failing to connect to the JIS Server. The default page is `ConnectionFailed.html` and it contains a list of the common reasons for this failure.
VersionMismatchPage	Defines an html page that is displayed in case of mismatch between the Client's code and the JIS Server's code. The default page is called `VersionMismatch.html`, and it contains a list of the common reasons for such an error.
LanguageNotSupported	Defines an html page that is displayed when the language used in the Application is not supported. The default page is called `Language NotSupported.html` and it contains a message saying that the JVM you are using does not support the Application's language.

Session Initialization

The following Session Initialization settings can be changed by the user:

Table 10. Session Initialization settings that can be changed by user

Setting	Description
RequestProfile	Before launching the session, allows the user to specify the profile of the session. When set to **TRUE**, prompts the profile dialog box where the user's profile must be inserted. When the setting is **FALSE** (default), the general `<ApplName>.ini` file is used, unless the "Profile" parameter is specified.
Profile	Determines the profile that is used. When the RequestProfile parameter is set to **TRUE**, initializes the session using the value displayed to the user. The default setting leaves the profile name field empty. When left empty, the `<ApplName>.ini` file is used.

Configurable User Messages

The following Configurable User Message settings can be changed by the user:

Table 11. Configurable User Message settings that can be changed by user

Setting	Description
ClientConnectionMessage	The following default message is displayed in the browser during the connection to the server: "Connecting to JIS Server". This message can be overridden by modifying the parameter's value.
ClientTerminationMessage	The following default message is displayed in the browser when the session has terminated: "Application terminated". This message can be overridden by modifying the parameter's value.

Scalability: Server Farm

The following Scalability settings can be changed by the user:

Table 12. Scalability settings that can be changed by user

Setting	Description
FullClientURL	In conjunction with `UseNewHTML=TRUE`, when downloading a new applet, specifies the URL address of the applet. When left empty, the current applet's URL is used (only the web server's name is replaced).
FullClientFrame	Defines the type of frame in which the redirected Web page is displayed. Possible values are: "`_self`" - show in the window (frame) that contains the applet. "`_parent`" - show in the applet's parent frame. If the applet has no parent frame, acts as in "`_self`". "`_top`" - show in the top level frame of the applet's window. If the applet's frame is the top-level frame, acts as in "`_self`". "`_blank`" - show in a new, unnamed top-level window. `Name` - show in a frame or window called `<WindowName>`.
UseNewHTML	When the client is redirected to another machine, defines whether to download a new applet based on the address given in the `FullClientURL` html parameter (**TRUE**), or connect directly to another server without downloading a new applet (**FALSE**).

Printing

The following Printing settings can be changed by the user:

Table 13. Printing settings that can be changed by user

Setting	Description
UseJavaPrintDialog	Enables printing via the Java Page Setup dialog box when a Java plug-in version 1.2 and higher is installed. From Java plug-in version 1.4 this is the default behavior. Setting this parameter to FALSE disables the feature and the operating system's Print dialog box is used.
PrintPageOrientation	Enables you to set the printed page's orientation to either Portrait or Landscape, when using a Java plug-in version 1.2 or higher. When set, this setting is then the default for all printing jobs.

Archive Files Provided by JIS

Server Side Archive Files

jacadasv.jar - This archive file contains the JIS Server classes.

Client Side Archives Files

clbase.jar This archive file contains the core classes of the Java client.

clbase-signed.jar A signed version of the clbase.jar archive file.

clhost.jar This archive file contains the classes required for viewing host screens on the client.

clprint.jar Contains the XML parser and all the printing emulation classes.

clfull-signed.jar This archive file exists only in a signed version. It contains:

- The core classes of the Java client.
- The classes required for viewing host screens on the client.
- The classes required for printing emulation.

Choose the archive file to use according to the features you wish to have available:

	Basic client	Screen emulation	Window printing	Printing emulation	System clipboard
clbase	x				
clbase & clhost	x	x			
clbase & clprint	x			x	
clbase-signed	x		x		x
clfull-signed	x	x	x	x	x

Note: The clhost archive cannot be used separately from the clbase archive.

Java Client Administrator Related Archive

jam.jar This archive file contains the JIS Administrator classes.

Note: The archive files are placed under: `<InstallDir>\classes\cst\`

Signed and Unsigned Archive Files

JIS class archives are available in either signed or unsigned versions. Due to security restrictions imposed by the Java Virtual Machine (JVM), some of the features JIS provides can only work using signed files. Following is the list of signed and unsigned archive files JIS provides:

Unsigned Archives	Signed Archives
clbase.jar	clbase-signed.jar
clhost.jar	
clprintjar	
jacstart.jar	
	clfull-signed.jar

Using Signed Files

Using signed files gives you the benefit of using the following features:

- Host-to-Client Printing
- Host screen and GUI window printing
- System clipboard activities (cut, paste) on non-Java programs
- Redirecting a client to other servers with a single class-download. See “Scalability” on page 89.

Note: Using a signed archive file prompts a security warning dialog box.

Sun Java Plugin prompts the following message when you download the signed archive:

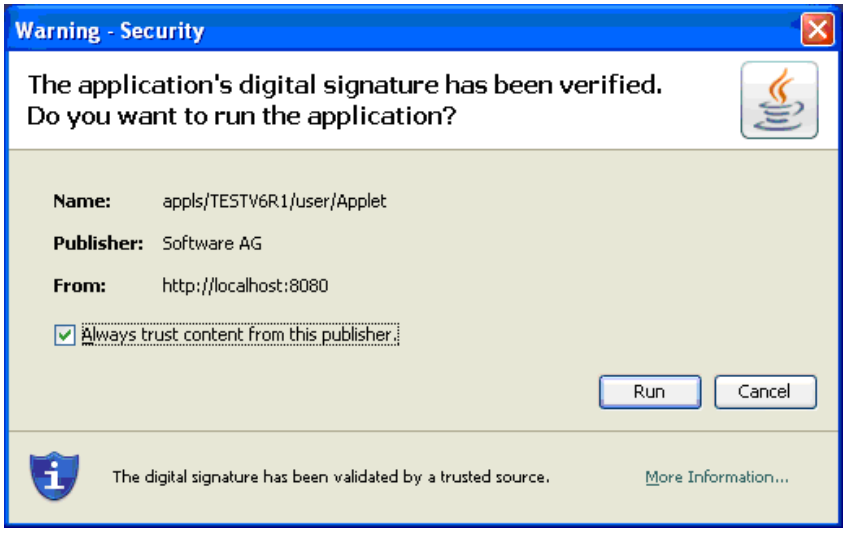


Figure 7. Plugin warning dialog box

The Application HTML Files Generated During Compilation

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

Table 14. Application HTML files generated during compilation

HTML File	Description
`<ApplName>.html`	Runs an unsigned client
`<ApplName>-signed.html`	Runs a signed client using signed archives
`<ApplName>-browser.html`	Runs the Java client inside the browser, not in a separate window. The applet opens in the top left-hand corner of the browser window. This option provides a more “web-like” look compared to the `<ApplName>.html` file.

Running the Java Client from an Applet Using a Launcher

The `cst.client.startup` package includes two different launchers which are used by the standard applet—as created by the compilation process:

- JacadaBasicLauncher – the standard launcher.
- JacadaLoginLauncher – a launcher that waits for the user to enter a profile name and click OK. It is used when the `RequestProfile` parameter in the application's HTML file is set to TRUE.

In addition, this package contains two other classes:

- JacadaLauncherInterface – an interface that must be implemented by all Java Client launchers. It defines a small set of services that the Application expects to receive from the launcher.
- JacadaLauncherCreator – a class that creates a tailored launcher, according to the parameter settings in the HTML file.

JacadaBasicLauncher

The basic launcher automatically starts the Application when its init method is called. This launcher only displays system messages.

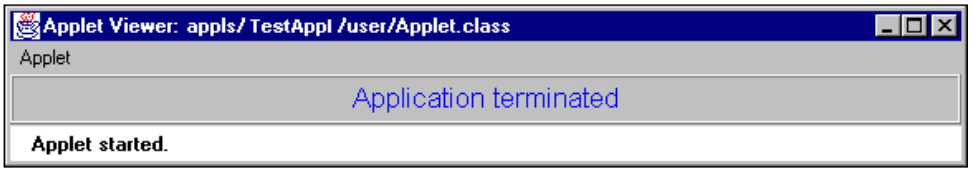


Figure 8. BasicLauncher

JacadaLoginLauncher

This launcher allows you to enter your user name (login) and start the Application using the personalized application setting. It shows messages from the Server in the lower half of its display. In addition, it allows you to restart the Application once it is finished.

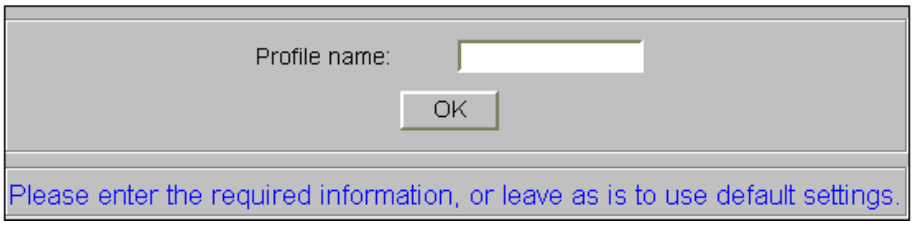


Figure 9. LoginLauncher

Running the Java Client from a Java Application

The Java Client can run from any Java application. It does not require an applet to be launched. This means that a Java client can either run as a standalone application, or be integrated with the code of other Java-based applications.

To run a standalone Java Client application, from the command prompt, use the `JAVA` command to run the class `appls.<ApplName>.user.Applet`.

The following requirements apply:

- You must include the directory `<InstallDir>\classes` in your `CLASSPATH`.
- You must also include one or more "Client Side Archive Files" in your `CLASSPATH`; for example:
  `<RootDir>\classes\cst\clbase.jar`
  The specific archive file or files that you choose depends on the features you wish to make available in your application. See "Client Side Archives Files" on page 59

The applet class contains a main () method that allows it to run as an application too.

The Application's Params File

The compilation process generates the `params` file. This file contains the default parameters and their settings. The `params` file supports, where relevant, the same parameters that are set for an applet in the html file.

The generated params file sits under:

`<RootDir>/classes/appls/<ApplName>/user/params.txt`.

In addition, the params file includes the following two parameters:

ApplicationClass=appls.<ApplName>.user.JacadaStarter - The JacadaStarter class that initializes the application. The setting indicates which application is initialized.

CodeBase= <FullyQualifiedURL> (default http://localhost/classes) - The setting indicates where the application's files that are loaded during runtime are stored.

Customized Application Settings

Application parameters setting may be set differently for different usage. Each customized parameters setting should be stored in a separate params file.

To create an additional params file:

1. Duplicate the default `params` file, using a different name.
2. Change the setting as desired.

To initialize an application with the desired params file, run the following class:

```
cst.client.startup.JacadaBasicLauncher
```

With the following argument:

```
appls/<ApplName>/user/<ParamsFileName>
```

Example 11. Customized application settings

```
java cst.client.startup.JacadaBasicLauncher  appls/mbf/user/params1.txt
```

The following requirements apply:

- You must include the directory `<RootDir>\classes` in your `CLASSPATH`.
- You must also include one or more "Client Side Archive Files" in your `CLASSPATH`; for example:
  `<RootDir>\classes\cst\clbase.jar`
  The specific archive file or files that you choose depends on the features you wish to make available in your application. See "Client Side Archives Files" on page 59

Running the Java Client Inside a Browser Window

The Java client can run inside a Browser window instead of in a separate window.

> **Note:** The popup windows and the host screen are displayed in separate windows.

To run the Java client inside a browser window:

1. Launch the `<ApplName>-browser.html` file. The client window is aligned with the upper left-hand corner of the browser window.

Alternatively, you can:

1. Add the following parameter to the application's `<ApplName>.html` file:

```
<PARAM name="RunInsideBrowser"      value="true">
```

2. Run your Application.

Fitting the Application Window into the Browser

This section applies to users of the `<ApplName>.html` file. The `<ApplName>-browser.html` file is automatically created with these modifications in place.

The space for an Application running within a browser is reduced by the space taken up by the browser itself. This must be taken into account when converting your application and setting up the runtime.

You can solve the problem of space by first reducing the application window's size and then fine-tuning it to fit the space available in the Browser's window.

To reduce the application window size:

Convert the application in a lower resolution than the screen resolution. For example, if your screen is set to a 1024x768 resolution, convert the application as 800x600.

To optimize the application window size within the browser:

- In the application HTML file, increase the HEIGHT and WIDTH parameter, to enlarge the application window:

Example12. Optimize the application window size within the browser

```
<APPLET…..
         WIDTH=760
         HEIGHT=450>
```

- In the application HTML file, decrease the applet's margins to zero:

```
<BODY TOPMARGIN=0 LEFTMARGIN=0 MARGINGEIGHT=0
 MARGINWIDTH=0>
```

Changing the Application's Look

The Application runs inside the browser using a default look. This includes a thin gray border around the application window, and white text on a blue background caption.

The customer can change this look using code extension. The compilation process creates a file called `SubApplContainer.java` in the directory `JacadaFiles/src/appls/<ApplName>/user`. This file implements the default look, but customers can change it in order to modify the look as they see fit.

Limitations

Since the Application runs in a browser it does not have its own menu bar. As a result, all the Application's commands must be made available as buttons or accelerators.

Note: When you create a menu bar, though it remains invisible, parts of it are accessible using the RMB popup menus, e.g. the List menu when right-clicking a table control.

Use Archives to Reduce Download Time of Java Classes

Application-specific Java classes are downloaded to the client one class at a time, and in an uncompressed format. Due to the inefficiency of the HTTP protocol, this process is time consuming. One way of reducing download time is by collecting the class files into one archive file. This way, all the application classes are downloaded to the client in one go.

JIS Interface Server provides you with a batch file that collects application-specific classes into JAR files. This batch file—`makejar.bat`—is included with JIS Interface Server and resides under the JIS Interface Server installation directory:

```
<InstallDir>\JacadaFiles\utils\makejar\makejar.bat
```

For more information about the `makejar.bat` utility, consult the `readme.txt` file that resides under the same directory.

Referencing Application Specific Archive in Application HTML File

Application specific classes that have been placed in archive files must be referenced in the application HTML file.

Referencing a JAR File in the Application HTML File

The following reference to the Client core JAR file is automatically added to the application html file during the compilation process:

```
ARCHIVE="cst/clbase.jar".
```

If you wish the browser to recognize other JAR files, put down their names, separated by a comma.

Example 13. Referencing a JAR File in the Application HTML File

If you have compressed the classes of an Application called `MyAppl` into a JAR file, add the file name as follows:

```
ARCHIVE="cst/clbase.jar,MyAppl.jar"
```

Improving Download Time of an Unsigned Application

This feature allows the downloading of host screens only if needed.

The classes required for viewing host screens on the client are contained in an archive file called `clhost.jar`

The clhost.jar archive file is downloaded to the client during initialization. If you have no need to view the host screen during runtime, you can choose not to load this archive file during initialization. This saves on download time.

Excluding the Downloading of the clhost.jar File

A reference to the `clhost.jar` file is added to the `<ApplName>.html` file during the compilation process:

```
ARCHIVE="cst/clbase.jar,cst/clhost.jar"
```

The reference to the `clhost.jar` file must be removed.

Optimizing Fonts for Non-Windows Platforms

The font settings you define in JIS Interface Server are best displayed on Windows-based platforms. When running an Application on a different platform a non-standard font may be displayed. This can have unexpected effects on the GUI display.

To avoid this from happening, you can control the way the font is displayed on the Java client by using the font substitution mechanism. This mechanism constitutes a substitution table that shows the original font setting beside the font setting you wish to display.

Example 14. Font substitution mechanism

```
Dialog-11=Dialog-20
```

Whenever the Java client is supposed to display Dialog font in 11 points, as defined in JIS Interface Server, it will display Dialog font, 20 points.

Note: You may add a font substitution resource file, containing a font substitution table to each Application.

The Font Substitution Resource File

This section deals with the location and contents of the font substitution resource file.

The Resource File Location and Name

Place the resource file under the following directory:

```
<InstallDir>\JacadaFiles\classes\cst\client\resources\<FileName>.res
```

File names are derived from the type of system running the client. If you have doubts as to the name of the file, run your application in Debug mode. The file name you should use appears in the JVM's Java console output.

The font substitution resource file name is a combination of the following factors

Combined, the file name looks as follows:
`fonts_<OS>_<Vendor>_<Version>.res`

The JDK version number is optional. When not including the JDK version number, the file name looks as follows: `fonts_<OS>_<Vendor>.res`

Example 15. File name examples

- When running the client on Windows XP with Sun's JRE 1.6.15, the file's name is:

  `fonts_Windows_XP_Sun_Microsystems_Inc_1.6.0_15.res`

  or:

  `fonts_Windows_XP_Sun_Microsystems_Inc.res`

Note: When both options exist, the version-specific file is read.

Contents of the Font Substitution Resource File

In the resource file create a list of the fonts' original settings and their substitutes. The syntax used for each line in this file is as follows:

```
FontName-[Style-]Size=NewFontName-[NewStyle-]NewSize
```

FontName	Dialog
	Times Roman
	Helvetica
	Courier
	ZapfDingbats

Style	bold italic bolditalic (Note that the Style setting is optional)
Size	Font size

Example 16. Font substitution

The following settings will effect the dialog box font in the described way:

Dialog-11=Dialog-20	Increases the size of Dialog font from 11 to 20 points
Dialog-11=Dialog-bold-20	Increases the size of Dialog font from 11 to 20 points and makes it bold
Dialog-11=Courier-italic-20	Uses Courier italic, 20 points, instead of plain Dialog, 11 points

Note: The font names in the substitution table are those used by the Java clients. Windows, and consequently JIS Interface Server, use different font names. The font sizes and types supported by Java are shown in “Java Client Limitations” on page 253.

Debugging the Font Substitution Mechanism

To run your Application in Debug mode, set the `DebugLevel` parameter to 10 in the application html file.

Example 17. Debugging the font substitution mechanism

```
<PARAM name = "DebugLevel"   value = "10">
```

Information Retrieval From Font Debugging

Debug the font substitution mechanism to retrieve the following information:

- The name of the substitution resource file

  When running in Debug mode, the name of the font resource file is printed to the log in the following manner: "Looking for fonts resource at: `<FileName>`"
- Font activity in the application

  The fonts' settings are also printed to the log file, in the following manner: "Using font: Dialog-20 (original font requested: Dialog-11)"—this entry shows the original GUI request: Dialog font, size 11, and the font that is displayed instead: Dialog font size 20. A font setting is logged on its first appearance in the Application.

Note: The debugging results are printed to the JVM's Java console output.

Chapter 3. Optimizing the JIS Server

The following topics are discussed:

- JIS Server *.ini File Settings - Provides you with a list of the JIS *.ini file settings. Many of the settings in the JIS *.ini file deal with aspects of connectivity between the JIS Server and the Client Application.
- Scalability - Provides you with information required to implement a scalable JIS Server system that dynamically balances the load in response to runtime demands.
- JIS Server Logging Support - Introduces the JIS Server mechanism for tracking and viewing the session status information for each process on your server system.
- JIS Administrator - Introduces the JIS Administrator system. The JIS Administrator's function is twofold: You use it to monitor the JIS Server and it serves as a utility to configure the runtime application.
- Running the JIS Server as a Windows Service - Describes two utilities you may use to install the JIS Server as a Windows service. A Windows service allows you to provide continuous, unattended access to your application without compromising the security of the server machine.
- Managing User Profiles - Each application has a runtime INI file in which its various settings are stored. The JIS Server allows for a separate runtime INI file to be assigned to each user.

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.

JIS Server *.ini File Settings

The JIS Server initialization file contains settings that deal, to a large extent, with aspects of connectivity between the JIS Server and the client Application. The settings are editable and their values can be changed to enhance the performance of the JIS Server.

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]`

`IniVersion`	Optional parameter. The developer can use this parameter to specify a character string that will appear in the debug log to identify the ini file being used.
`KeepAliveTimeout`	Time-out for sub-processes to send a keep-alive notification to the main process. If the main process did not receive a keep-alive notification until the time-out expires, the reference to the sub-process is removed. The value is calculated as multiples of 'KeepAliveTimerTick'. The default value is 10 (x 'KeepAliveTimerTick').
`KeepAliveTimerTick`	Interval at which sub-processes send a keep-alive notification to the main process. Default value is 60 seconds.
`MaxMachine-Applications`	Maximum number of applications to be run concurrently on the server machine.
`MaxProcess-Applications`	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 will look for a free port, within the port range, before giving up. Default value: 2.
`RegistryPortRange`	Range of port allocations for the RMI Registry (minimum of one port is needed). Format: `<LowPort> - <HighPort>` Example: 1900-1901 Default port number is 2100.
`RegistrySpawn Timeout`	The time in milliseconds the server waits for the NodeRegistry process to start. The default is 45,000 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.
`RMISocketTimeout`	Defines the time in seconds that a RMI client waits for a server to respond. When a remote method is called on a remote object it will time-out after the defined time, throwing a 'RemoteException'. Default value: 20 seconds.

`RtDebugFileMaxSize`	Specifies the maximum size of the server log file in bytes, if desired. **Example**: RtDebugFileMaxSize=1000000 Limit the 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 42.) If the parameter is specified in both places, the value specified on the command line takes precedence.
`RtDebugMaxFiles`	Specifies the number of revisions of server log files to keep. 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 42.) 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=ACCESS,METHOD 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 110 for more information. Debug filters can also be specified on the JIS Server command line (See “The JIS Server Command Line Parameters” on page 42.) 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 110 for more information.

`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. A debug level of “1” results in few log records being written.

The debug level can also be specified on the JIS Server command line (See “The JIS Server Command Line Parameters” on page 42.) 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 GmbH recommends that debug level larger than 70 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.

`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 42.)

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 process, a numeric suffix is automatically affixed so that the file name becomes `Debug_1.0.log`, for example.

`RtRootDir`	The runtime root directory. For example: `<InstallDir>\JacadaFiles`. Note that the string "$RootDir" in settings that include it in their path will be replaced in runtime with the root directory defined in this setting.
`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: `<LowPort> - <HighPort>` Example: `1900-1901` Default value: `1100-1100`
`ServerPortRange2`	Optional range for the secondary communication port number. Format: `<LowPort> - <HighPort>` Example: `1900-1901` Default is the port range defined in `ServerPortRange`.
`SocketImplFactory`	The fully qualified classname of a custom implementation of the socket factory class on the server side. Example: `appls.applname.server.user.MySocketFactoryImp` When undefined, uses regular Java sockets.

`SoftLimitMargin-Percent`	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, will 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 all other processes.
`StartScanAtRandomPort`	To avoid a situation in which all processes vie for the same free port, this setting defines for each process a random starting point within the range of allocated ports. If the port range is 2000-3000 and the random starting point is 2500, then the process will start looking for the free port at port 2500-3000, and if not found, from port 2000-2499.
`StdoutEncoding`	Defines which Encoding setting the Server processes use when reading the output of spawned processes. This is dependent on the iSeries operating system. For example, OS V4R2 requires the following setting: `Stdout=Cp037`
`SystemConnection-Timeout`	On startup, the interval in seconds in which the main node tries to establish a connection with a root node on another machine. Default is 120 seconds.
`WaitForSpawned`	Maximum waiting time for a server process to start and initialize. After this interval the spawned process is considered to have failed. The default value is 60 seconds.

[HTTP]

`HTTPPortRange`	Specifies the port range of the JIS Server's HTTP connection. Specify as two ports separated by a dash, low port first. Default is 8080-8080
`ResourceBase`	Specifies the base directory from which the location of image files is defined. Usually, this should be the runtime root directory. For example: `C:/`JIS Interface Server/`JacadaFiles`

[Applications]

`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.

[Application_Name]

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

`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>\classes\appls\<ApplName>\server\resources`

[VMCommandLine]

`Classpath`	The classpath used by the Java VM for spawned processes. **Note**: The '`-classpath`' command line option (or its equivalent in accordance with the VM implementation in use) has to be prefixed to the parameter value.
`JavaMemory`	Subset of the Java VM specific command line options used for memory settings. **Note**: If not set, the Java VM uses its own defaults. Example: -ms10m -mx50m
`JavaOptions`	Java VM specific command line options. **Note**: If not set, the Java VM uses its own defaults. Example: -`Djava.security.policy=<RootDir>\security\jacadasv.policy`
`JavaVM`	External Java 21 executable for spawned server processes. Example: <JavaHome>/bin/java

[ServerMachines]

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

[Sessions]

`MaxProcessSessions`	Maximum sessions to be allocated in a server process. Default is 1000.
`MaxMachineSessions`	Maximum sessions to be allocated in a server machine. Default is 1000.

`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 the '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 specified by this parameter are allocated below the margin defined in the 'SoftLimitMarginPercent' parameter.

[SessionTimeouts]

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

`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.

[ScoreWeights]

These parameters are used in load balancing and scalability calculations.

`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.
`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.

[LogClasses]

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

`<LogJavaClassName>`	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 just gives you the ability to use the log class. To actually enable the log class, you must create a `[<LogJavaClassName>]` section for the class in question, and include the Enable parameter in that section.
	
	Format: `<JavaLogClass>=<value>`.
	
	`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 `[<LogJavaClassName>]` section. The four possible log classes are `SessionLog`, `SessionCountLog`, `XMLLog`, and `XMLServer`. The table below describes the log classes.

`SessionLog`	The SessionLog writes the session status information to a log file in the format of a comma delimited list. The SessionLog contains information For each active session 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 at any time.
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.

`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.

`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. Between writes, 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` JIS automatically inserts a timestamp in the file name. For example, if you specify `Session.log` as the filename 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.
`FileInterval`	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 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.

`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` 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.

`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.

`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.

`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

Scalability provides you with a mechanism for creating a JIS Server system that can comprise a single server computer or a bank 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 125. 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 10 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.

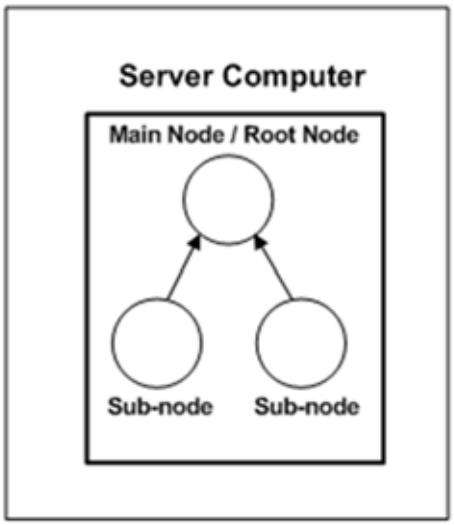


Figure 10. A Single Server

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 11, 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.

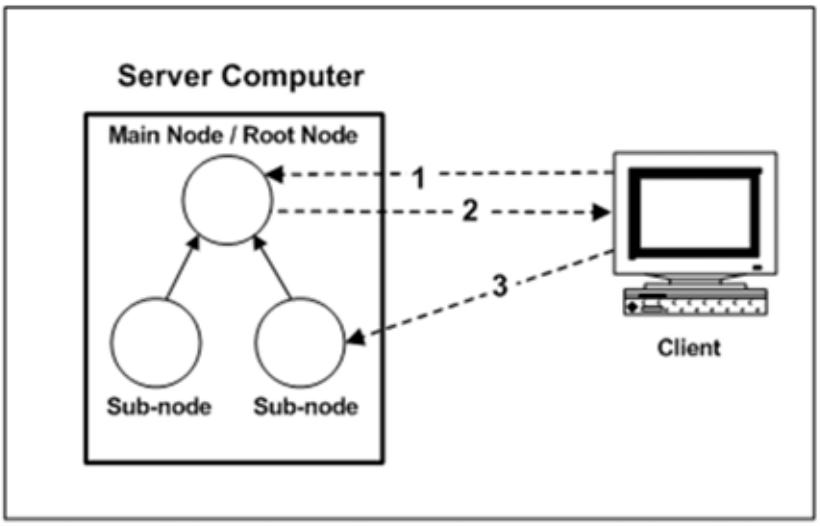


Figure 11. Control Flow When Starting a Session

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 95.

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.

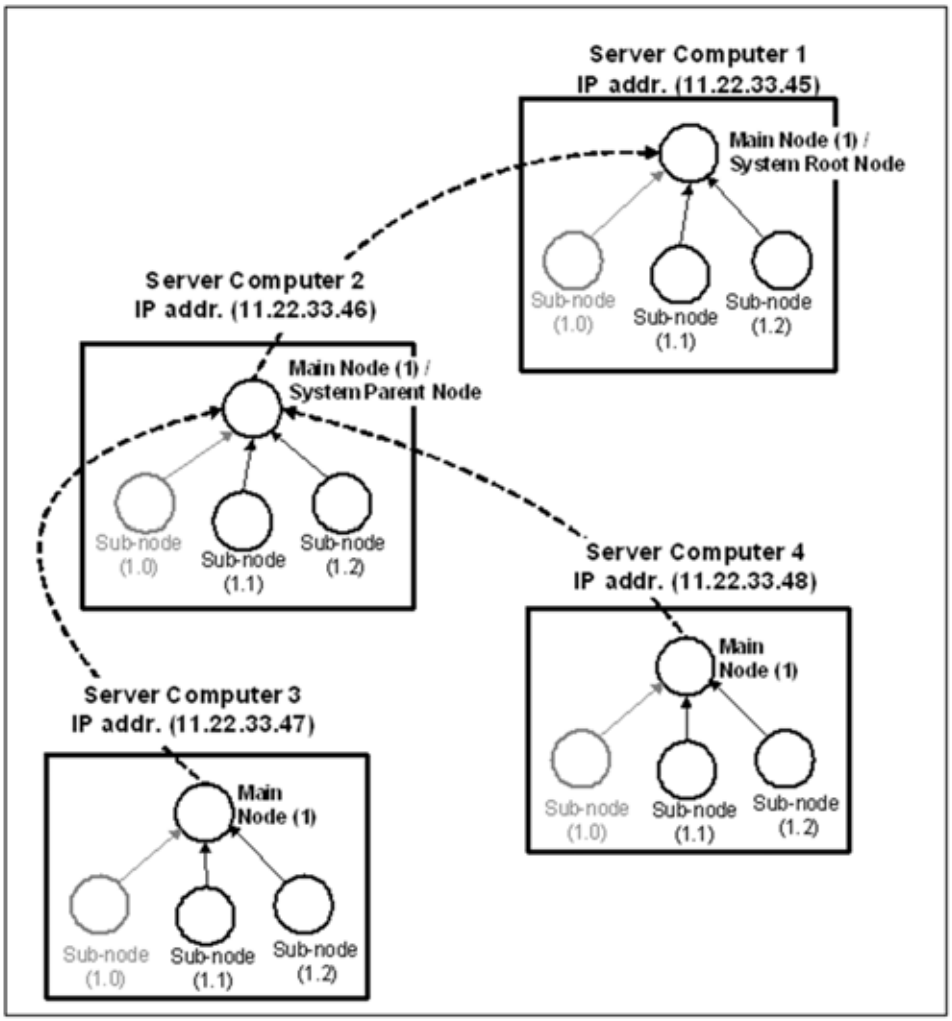


Figure 12. A Server Farm

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 12.

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 12, 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 12 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 12, 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.

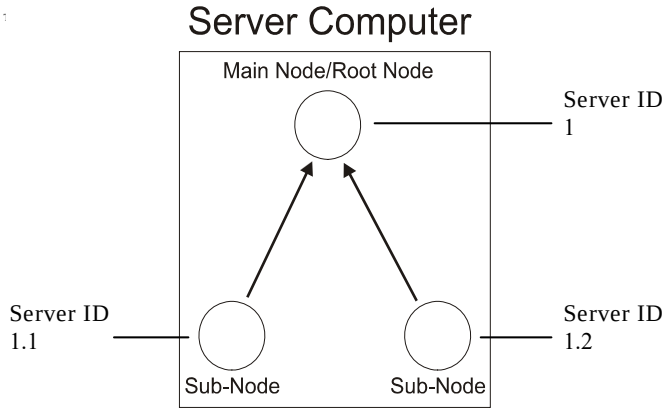


Figure 13. Identifying Server Modules

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 12, 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 GmbH 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 GmbH 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 74.

Targeting ini Parameters to a Particular Machine or Node Level

It is possible to selectively target the parameters in an 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 `jacada.ini` file shown in Example 20, "Example of jacadasv.ini for a multiple server-computer system", which begins on page 100, 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 18. Example of jacadasv.ini for a multiple server-computer environment

In a multiple server-computer environment, you could have a `jacada.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 19. Example of jacadasv.ini for a single server-computer system

This example provides the jacadasv.ini settings and parameters that you may expect to see in 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 is 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 parameter
// 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 20. Example of jacadasv.ini for a multiple server-computer system

This example provides the jacadasv.ini settings and parameters that you may expect to see in 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 is 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\
```

HTTP/S Communication

The default for Java clients is to use direct port connections to connect to the JIS Server. However, when the Server is placed behind a firewall, in a secured network, direct access is usually banned. In such a case, we use the HTTP/S communication as a bridge between the clients and the server, and wrap our protocol with HTTP requests, since HTTP connections are usually available.

Note: JIS 9.0.4 introduces the option of running the HTTP/S communication as part of the JIS standalone server. For more information refer to the JIS 9.0.4 release notes.

JIS Server Logging Support

The JIS Server provides a 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.

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, the Session Log, the XMLLog, and the XMLServer.

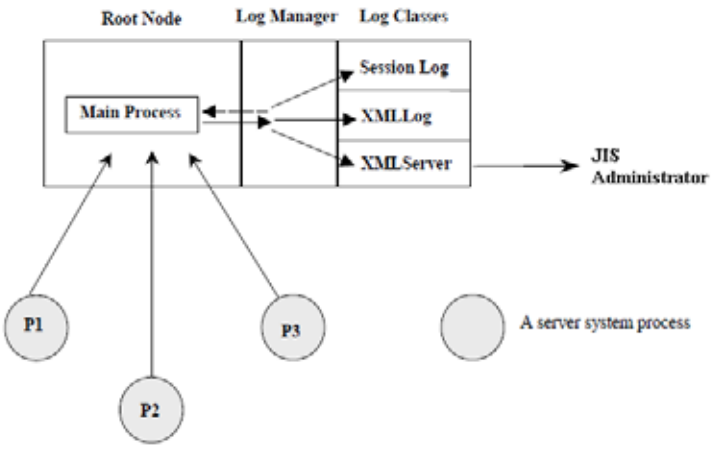


Figure 14. JIS Server logging architecture

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

Table 15. Server System's logging support components (Sheet 1 of 2)

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.

Table 15. Server System's logging support components (Sheet 2 of 2)

Component	Function
`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 the root node is dependent on the Log class performing the request.

The Server System Log Classes

This section describes the Server system log classes in detail.

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 for 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.

The following table indicates the record parameters written to the SessionLog and provides a description of each parameter:

Table 16. Record parameters written to the SessionLog (Sheet 1 of 2)

Parameter	Description
`Server Process ID`	The server IP address and Port1. Separated by a colon (:). `192.90.14.4: 1100`
`Session ID`	A numeral representing the sessions position on the process relative to any other session.
`Time Connected`	A time stamp indicating the date and time that the session connected to the process. `2/18/99 12:19:40`

Table 16. Record parameters written to the SessionLog (Sheet 2 of 2)

Parameter	Description
`Time Disconnected`	A time stamp indicating the date and time that the session was disconnected from the process. `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 Name`	The name of the application.
`Last Transaction`	The time of the last recorded server activity in the current session.
`Total Duration`	Time taken from request reception to the return of a response.
`Total Transactions`	The total number of transactions performed by the process.
`Net Avg Duration`	The average transaction duration - calculated by dividing `Total Transaction Duration` by `Number of Transactions`.
`Current Screen`	Provides the name of the current Subapplication.
`Last 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` file. 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.

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 17. Setting LogClasses and their jacadasv.ini 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.
`TimerTick`	Defines the amount of time between successive log class request 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. 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 example LogClasses section below requests that only a SessionLog be produced; the XMLLog and XMLServer log are not created.

```
[LogClasses]
;XMLLog=
XMLServer=0
SessionLog=1
```

> **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<Directory path>`.

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`

If there is more than one server process, a numeric suffix is automatically affixed so that the file name becomes `Debug_1.0.log`.

Example 21. Server log file

`jacadasv.bat` **-d50 -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, the log file is renamed and a new log file is started.

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`.

Debug Filters

Debug Filters are tools to help you accomplish specific types of logging. They are implemented by using the command line parameter
`-f<DebugLogFilters(colon delimited)>`

We recommend that the Debug Filters be used with a low debug level; this makes it easy to find the filtered messages in the output. No filtered messages are printed, however, when the debug level is set to 0.

Example 22. Debug filters

```
-fPROFILING -d1
```

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 23. Addfilter

```
addfilter PROFILING
```

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.
- Host screen image.

Method Debug Filter

Filter name: METHOD

The method debug filter limits output to the method debug messages. See “Setting Runtime Generation Options” on page 22 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

The JIS Server and client log files include details about termination of the runtime — be it regular termination by the end user, or abnormal termination of the client or server. However, the server and client logging feature can be disabled by changing the debug level to 0. To provide a means of analyzing abnormal runtime termination, specific information about the process or 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 18. Information included in dump files (Sheet 1 of 2)

Level of Information	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.

Table 18. Information included in dump files (Sheet 2 of 2)

Level of Information	Description
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

The dump file name and location are as follows:

- In the Java client, the dump file is written to the Java console.
- In the server, 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 the `<ApplName>.ini` file and in the `jacadasv.ini` file. Set the following parameters to enable dump file generation and determine the various logging properties:

[SessionCoreDump]

Table 19. SessionCoreDump 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.

[HTML]

Table 20. HTML parameters

Parameter	Value	Description
AlwaysDump	0	Default value. Dump information is not added to the Java console.
	1	Always adds dump information to the Java console, regardless of how the client session ended, and regardless of the debug level.

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 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.

Client Core Dump File

The Client Core Dump file includes the following details:

Table 21. Client core dump file examples (Sheet 1 of 3)

Log Section	Information Type	Example
General Process Informatio n	Java vendor and version number	Java Version = 1.7.0_03 Java VM version = 22.1-b02 Java VM name = Java HotSpot(TM) 64-Bit Server VM Java Vendor = Oracle Corporation Java VM vendor = Oracle Corporation
	OS name and architecture number	Java Architecture = amd64 Java Data Model = 64 Operating system = Windows 7
	JIS version or PTF name	Version/PTF Name = 9.1.1
	JIS version Build number	BuildNumber = 9,0,0,561
	Machine IP address	Machine IP address: 12.34.56.789
	JVM memory consumption	Memory: Free: 49 Kb (3%) Total: 1488Kb
	Session termination details	Session was terminated due to: 09:26:33 22/10/12 Quit reason: Host communication failed
	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 21. Client core dump file examples (Sheet 2 of 3)

Log Section	Information Type	Example
Session Level Informatio n	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.
	Outgoing requests	`2/12/03 11:35:50 AM Client ->Server: Command hComp=0, id=22, action=8`

Table 21. Client core dump file examples (Sheet 3 of 3)

Log Section	Information Type	Example
	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.

Exception Informatio n	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 22. Session core dump file examples (Sheet 1 of 3)

Log Section	Information Type	Example
General Process Informatio n	Java vendor and version number	Java Version = 1.7.0_03 Java VM version = 22.1-b02 Java VM name = Java HotSpot(TM) 64-Bit Server VM Java Vendor = Oracle Corporation Java VM vendor = Oracle Corporation
	OS name and architecture number	Java Architecture = amd64 Java Data Model = 64 Operating system = Windows 7
	JIS version or PTF name	Version/PTF Name = 9.1.1
	JIS version Build number	BuildNumber = 9,0,0,561
	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 Level Informatio n	Session details	• Session ID • Application name • Application directory • Connection settings

Table 22. Session core dump file examples (Sheet 2 of 3)

Log Section	Information Type	Example
	Recent Subapplications	
	Emulator events	
	CreateFrame messages	
	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.
	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.

Table 22. Session core dump file examples (Sheet 3 of 3)

Log Section	Information Type	Example
	Cycle number	Information about internal data structure.
	Chronological order of events	A list of all the session's events in chronological order.
Exception Informatio n	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.ja va:112) at cst.server.applicat.MainSub Application.window Destroyed(MainSub Application.java:60)

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 may prevent the server from functioning properly.

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` file 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 and configuration warnings 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.

Reported Errors

Table 23. Reported errors (Sheet 1 of 2)

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 = 1`. The https test is only executed if `SupportHTTPS = 1`.

Table 23. Reported errors (Sheet 2 of 2)

Configuration Element	Possible Error
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.

Reported Warnings

Table 24. Reported 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 124 for a list of valid ranges.
Unrecognized Properties	The server does not recognize known properties. This 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

This section deals with the enabling of 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 at debug level 50.

In Server Mode, the Server Configuration Checker writes the configuration information to:

Log Name	`debug_1.log`
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.log`

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 42 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 25. Range of valid numeric properties in the jacadasv.ini (Sheet 1 of 3)

Property Name	Description	Default Value (In Seconds)	Valid Range
`KeepAliveTimerTick`	Time interval for sending keep alive to parent process.	60	10-600
`KeepAliveTimeout`	Process inactivity time-out.	10	1-100
`SystemConnection TimeOut`	Time interval for establishing connection between machines.	120	20-1200
`RegistrySpawn TimeOut`	Time-out for spawning the registry.	45	10-450

Table 25. Range of valid numeric properties in the jacadasv.ini (Sheet 2 of 3)

Property Name	Description	Default Value (In Seconds)	Valid Range
`WaitForSpawned`	Time interval for waiting for a spawned process.	60	20-600
`WaitForStatus`	Time interval for session allocation.	100ms	10-1000
`AllocLockTimeout`	Time-out for process lock during session allocation.	10	1-600
`ExpectingSession Timeout`	Time-out for client redirection.	300	10-3000
`SessionIdleTimeout`	Idle time-out for XML sessions.	3600	0-86400
`RMISocketTimeout`	Time-out for RMI calls.	20	10-200
`MsgboxTimeout`	Time-out for message box reply.	36000	3600-360000
`PanelTimeout`	Time-out for next panel selection.	36000	3600-360000
`GetTextFromUser Timeout`	Time-out for getting user text.	36000	3600-360000
`KeepAlive`	Client keep alive.	240	0-2400
`RecvTimeout`	Time-out for client reply.	200	60-2000
`ResponseTimeout`	Http response time-out.	100	20-1000

Table 25. Range of valid numeric properties in the jacadasv.ini (Sheet 3 of 3)

Property Name	Description	Default Value (In Seconds)	Valid Range
`SessionInitializationTimeoutInSeconds`	Time-out for preloaded session initialization.	300	10-3000
`MaxNewSessionsPerSecond`	Maximum number of pool sessions that can be created per second.	5	1-30
`PoolCreationDelayInSeconds`	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
`ValidityCheckIntervalInSeconds`	Frequency of the validity checks.	300	10-3600
`SizeCheckIntervalInSeconds`	Frequency of the checks whether the pool should be refilled.	10	1-3600
`StartUpSessionsPercent`	Percent of sessions supported in server start.	0	0-100
`SpareSessionsPercent`	Percent of spare sessions in ongoing state.	0	0-100

JIS Administrator

The JIS Administrator (JAM) provides the system administrator with a means of viewing and manipulating the JIS Server.

You can view the JIS Server system structure, as well as the process and session activity taking place on the server system.

You can manipulate the JIS Server as follows:

- Pause and resume its activity.

  -OR-
- Stop its execution altogether.

Note: JIS 9.0.4 introduces an option for running the JIS Administrator as a web application using the proprietary server. For more information see the product release notes.

Starting the JIS Administrator Command Line Utility

You can start the JIS Administrator from either the Server Machine, where the server machine is running, or remotely from a Windows workstation mapped to the Server Machine. Use the latter case, to run JIS Administrator from a remote site, or when the server is not installed on Windows.

Starting JIS Server 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 opens without displaying any contents. If the server is running then when the interface opens, it displays the server's status.

Note: Make sure -p http://localhost:8080/port is present in the Jam.bat command line. You can replace //localhost:8080/ with the address and port of the monitored server.

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 toolbar.
  -or-
- From the Connect menu select Connect Online.
  The Connect Online dialog box opens:

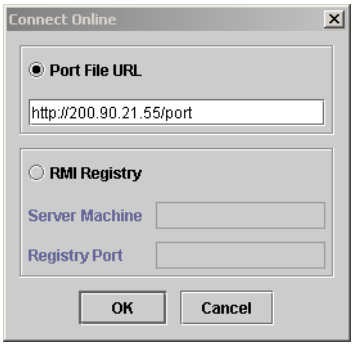


Figure 15. Connect Online dialog box

2 Choose either:

- The Port File URL option.
  -or-
- The RMI Registry option.

3 Enter the appropriate information as indicated in Table 26, "Parameters Used in the Connect Online Dialog Box".

Table 26. Parameters Used in the Connect Online Dialog Box

Parameter	Description
Port File URL	Enter the complete URL as indicated in the illustration. Make sure to include the: • IP address and port for the server machine http web server address • followed by **/port**.

Table 26. Parameters Used in the Connect Online Dialog Box

Parameter	Description
RMI Registry	When using the RMI registry you must supply the: • Server Machine's IP address. • Registry Port as indicated in the `jacadasv.ini`.
Server Machine	Enter the server machine's IP address in the text box.
Registry Port	The default registry port is **2100**. 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.

Debugging the JIS Administrator

The JAM 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 to the JIS Administrator parameter line any of the following switches:

-d Debug level. For example **`-d 50`**

-l Debug logging file directory. For 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.

The following is an example of the JIS Administrator startup command:

```
jam.bat -d 50 -l c:\temp
```

Note: Do not forget to leave a space between the parameter and its value.

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.

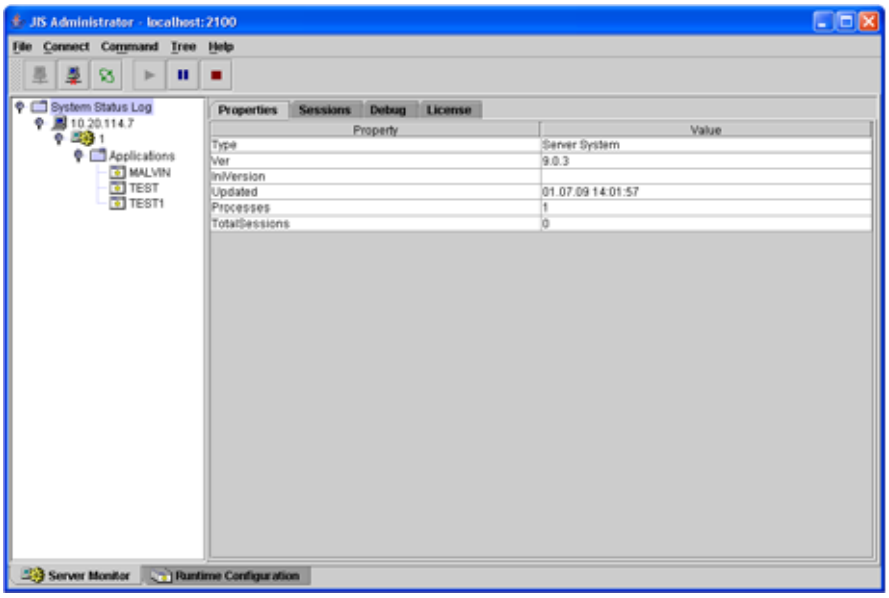


Figure 16. Server monitor interface

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. A table detailing the Properties and a description of their values is listed later in the chapter.

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 alpha-numerically sorted from top to bottom or from bottom to top.

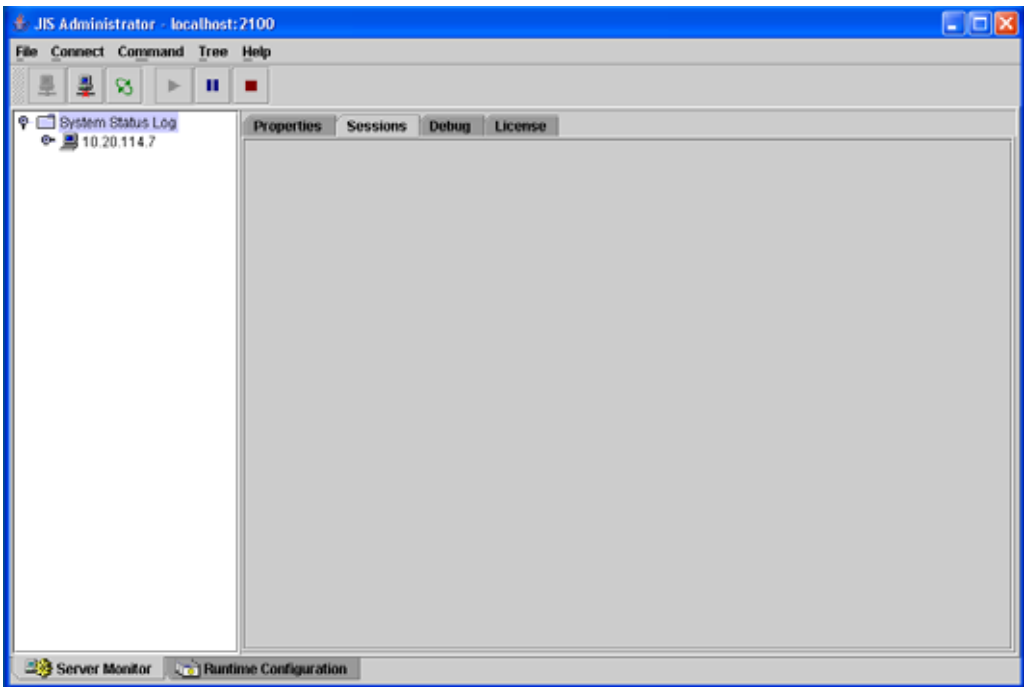


Figure 17. Sessions tab

The Sessions tab contains the following fields:

- Server
- ID
- Created
- User
- UserAddress
- ApplicationName
- LibraryName
- CurrentScreen
- State
- Transactions
- TotalDuration
- AvgDuration
- LastTransaction
- LastTransactionDuration

See the Session element section in “The Properties and Sessions Tabs” on page 132, for a description of these fields.

The Properties and Sessions Tabs

The properties in the Properties and Sessions tabs are described in Table 27. The properties and values listed on the Properties tab vary, according to the element in the System Status Log pane that is in focus.

Table 27. 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`	JIS version release number.
	`IniVersion`	Lists the IniVersion, if specified in the *.ini file.

Table 27. Properties in the Properties and Sessions tabs (Sheet 2 of 4)

System Status Log Pane Element	Property Name	Value Description
	`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.
	`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 that the process uses.
	`HttpPort`	The HTTP port number.
	`HttpsPort`	The HTTPS port number.

Table 27. Properties in the Properties and Sessions tabs (Sheet 3 of 4)

System Status Log Pane Element	Property Name	Value Description
	`State`	Indicates the activity state of the process. Possible values: INITIALIZING, STARTED, PAUSED, STOPPED, PENDING_START, PENDING_STOP, FAILED
Applications		
	`Applications`	Number of different applications running on the process.
<ApplName>		
	`Name`	Application Name.
	`TotalSessions`	Number of sessions on the process that are being used by the application.
	`TotalPenalty`	Total penalty ranking experienced by the process.
Session Folder		
	`Spare`	The number of free sessions available for use on the process.
	`Size`	The number of session in use on the process.
Specific Session		
	`ID`	Unique numeric ID for the Session, since the server was started.

Table 27. Properties in the Properties and Sessions tabs (Sheet 4 of 4)

System Status Log Pane Element	Property Name	Value Description
	`Created`	Date and time the session was 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 the screen. See JIS Server logging support for more details.
	`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 number of Debug Log files 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 18. The fields in the Debug tab are described in Table 28 on page 137.

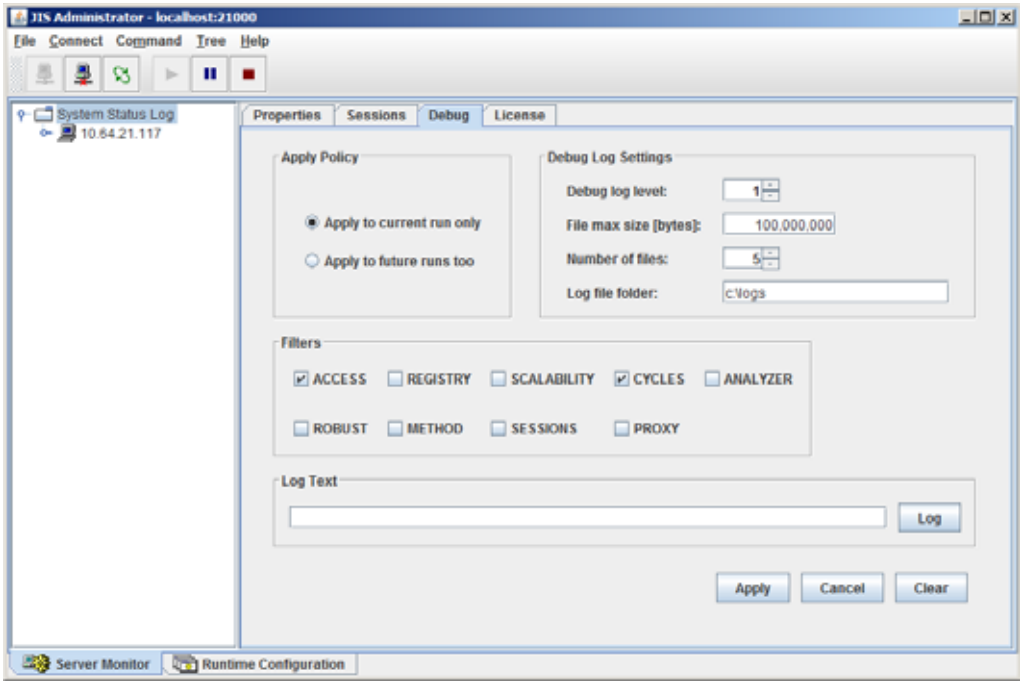


Figure 18. The Debug tab in the JIS Administrator

Table 28. 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, which is described in “RtDebugLevel” on page 77.
Log File Size	Corresponds to the `RtDebugFileMaxSize` INI file setting, which is described in “RtDebugFileMaxSize” on page 76“.
Number of Log Files	Corresponds to the `RtDebugMaxFiles` INI file setting, which is described in “RtDebugMaxFiles” on page 76.
Log Directory	Corresponds to the `RtLogsDir` INI file setting, which is described in “RtLogsDir” on page 77.

Table 28. 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 110.
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	Cancels 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.

The License Tab

The License tab displays runtime license details. To update a JIS 9.3 license, click Browse & Replace License and select the signed Software AG XML file. The file chooser remembers the last folder. Figure 19 shows the current UI with illustrative values.

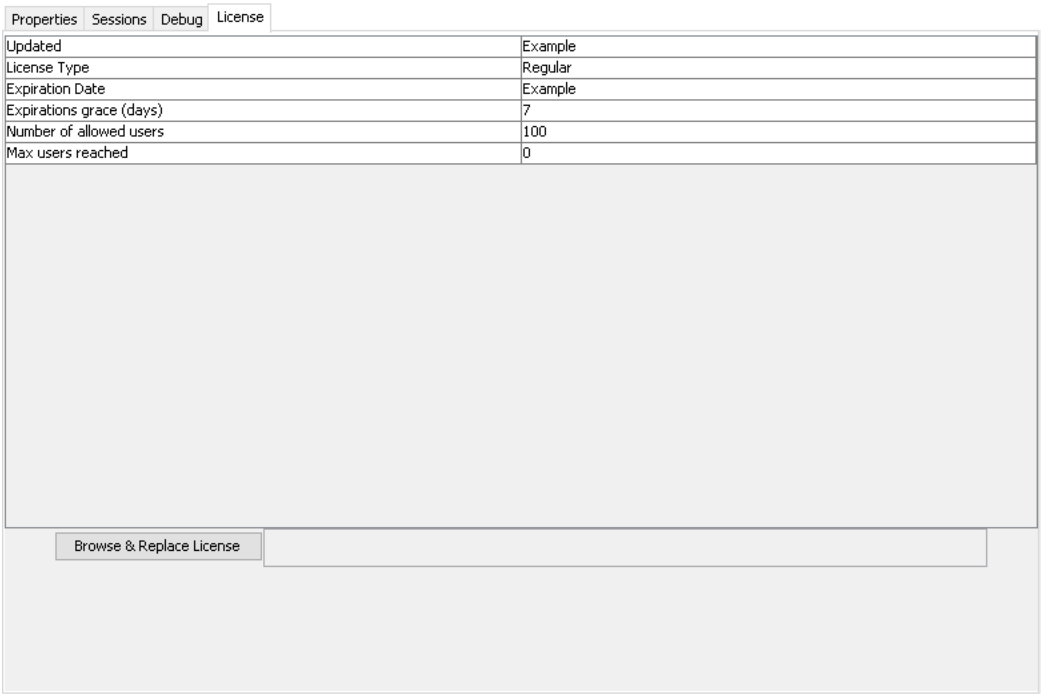


Figure 19. The License tab in the JIS Administrator

Table 29. Elements in the JIS Administrator License tab (Sheet 1 of 2)

License Tab Element	Description
Updated	The date the product license was last updated. Format is mm/dd/yy hh:mm:ss [AM|PM]

Table 29. Elements in the JIS Administrator License tab (Sheet 2 of 2)

License Tab Element	Description
License Type	Possible license types are: • Regular - sets limit on 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 limited by number of simultaneous users. • Unlimited
Expiration date	Date license expires. Contact your Software GmbH representative well before the expiration date to arrange for a license update.
Expiration grace	Additional days after the formal expiration date during which the license will continue to be valid.
Max users reached	Highest number of maximum simultaneous users seen by the Server during the current Server execution.
Browse & Replace License	Click Browse & Replace License and select the signed XML license file. The license is stored as SAGlicense.xml; license.dat points to it. The path field is read-only.

Operations you Perform Using the Server Monitor

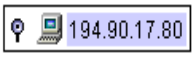
The most typical use of the Server monitor is to manipulate the JIS Server and to view the distribution of Server machines, applications, processes and sessions on the server system. Under certain circumstances your needs may require you to:

- Pause/Resume/Stop the JIS Server.
- Save/Open the System Status Log to file.
- Close sessions.

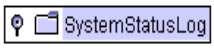
You can use the JIS Administrator to Pause, Resume or Stop the JIS Server. Each of these activities can be performed on either a single machine or on the entire farm.

> **Note:** When you pause or stop a single machine, the JIS Server remains active for other machines in the server farm.

You can either:

- Click the machine button 

and choose the action you wish to perform on a single machine.

  -or-
- Click the SystemStatusLog button 

and choose the action you wish to perform on a the entire 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 button or on the SystemStatusLog button.
2. Either:
   - Click the button on the tool bar.

     -or-
   - From the Command menu select Pause.

> **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 will allow new sessions to connect.

To resume the JIS Server:

1. Stand on the machine button or on the SystemStatusLog button.
2. Either:
   - Click the button on the tool bar.

     -or-
   - From the Command menu select Resume.

Note: Once the status of the Server changes to Resume, 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 the `Pending_Stop` status depends on the setting defined in the Stop JIS Server dialog box. See below.

To stop the JIS Server:

1. Stand on the Machine button or on the SystemStatusLog button.
2. Either:
   - Click the button on the tool bar.
     -or-
   - From the Command menu select Stop.

The Stop JIS Server dialog box opens:

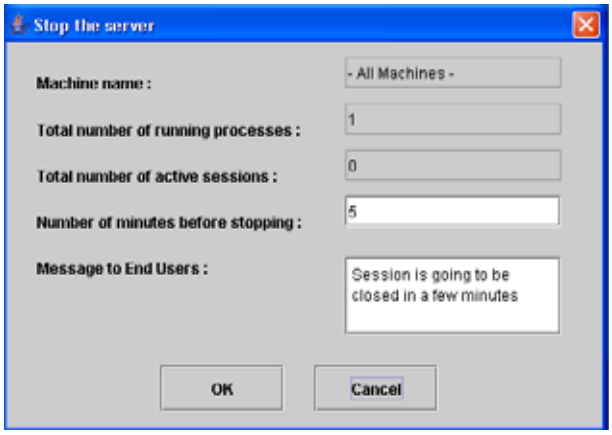


Figure 20. Stop JIS Server dialog box

The following parameters can be found in the Stop JIS Server dialog box:

Table 30. Stop JIS Server parameters (Sheet 1 of 2)

Parameter	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.

Table 30. Stop JIS Server parameters (Sheet 2 of 2)

Parameter	Description
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	In this editable field, determine the number of minutes the server will be in Pending_Stop status before stopping.
Message to End Users	In this editable field, define the message to be displayed to the end user when stopping the JIS Server.

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 and JIS Server Logging Support.

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. JAM 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.

3 Enter the saved log file name in the File Name edit box.
4 Click Open.

The log file is opened and the System Status Log is displayed in JAM.

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.
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 Session.
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

The behavior of the runtime environment is independent of a 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 may, if you wish, change these values and the new values are automatically preserved between runtime sessions.

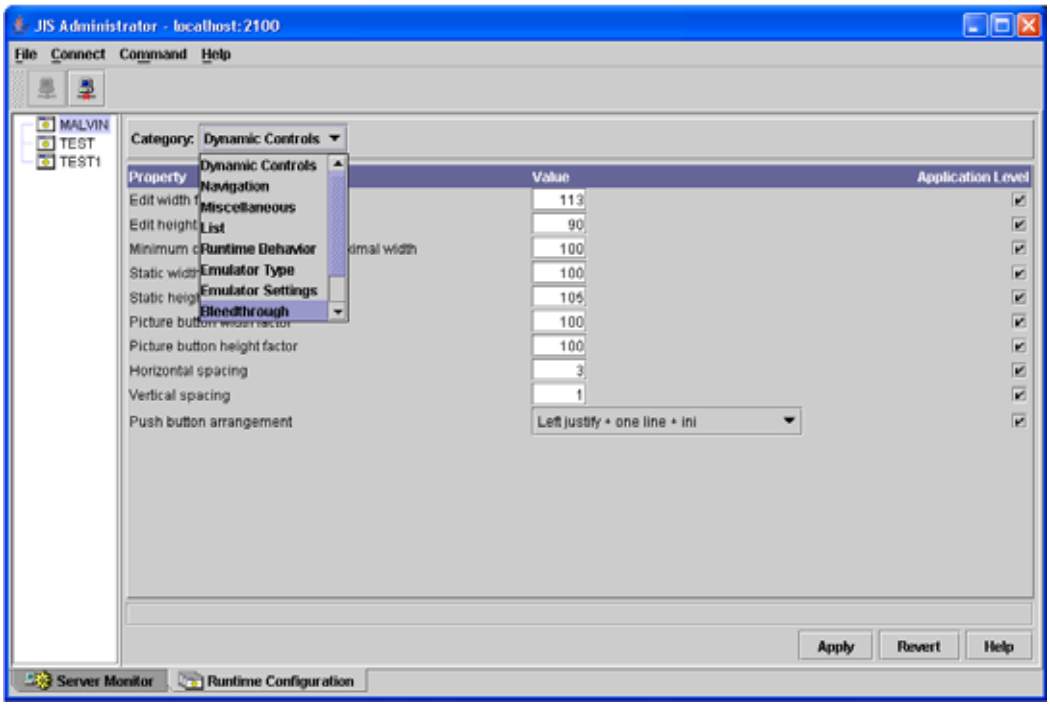


Figure 21. Runtime Configuration interface

The Runtime Configuration interface provides you with an option to implement these changes:

Table 31. Runtime Configuration interface parameters (Sheet 1 of 2)

Parameter	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 31. Runtime Configuration interface parameters (Sheet 2 of 2)

Parameter	Description
Category	The runtime options are grouped into logical categories. The categories are listed within a combo box labeled **Category**.
Property	The column lists all the parameter names included within a specific category.
Value	In this column, insert parameter values, or change existing ones.
Level	Depending upon whether you are running an application or a User Profile this column's heading will read Application Level or User Profile Level respectively. Accordingly, setting a parameter's check box determines the level on which the changes will take place. For more information, see section *Application level vs. User Profile level*.
Apply	Automatically writes the changes you have made to the runtime application ini file.
Revert	Reverts the changes you have made since the last Apply.

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 will affect all users of the specific application or library; changes on a User Profile level will 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 JIS Interface Server.

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, set the parameter you wish to change. The following message appears “Do you want to explicitly set the current value for this Application Level?” Click Yes.
3 In the Value column, set the value you wish to change.

Note: In most cases changes you make will not affect existing sessions. Some changes will take effect for new sessions while other changes will only take effect after server restart.

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, set 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.

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 start the server. 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.

Invoke `JBSToService.exe` accepts a variety of optional command line parameters by which you define the service's behavior. `JBSToService.exe` from the command line of a Windows runtime installation (not from the development kit)..

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>
        -n<Displayed name of the service>
        -x<Full path to the executable file>
        -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>
```

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"
JBSToService.exe -c -iJac -a -h"Second description" -n"JIS Service"
```

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.

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"`

> **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
```

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 "validity checks" on the values you choose. Using the `JBSToService.exe` utility carelessly could result in a damage to the operating system.

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.

Log File

The log file of `JBSService.exe`, which is created in <runtime installation root>\classes\logs\JBSService.log, 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").

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
```

> **Note:** Failing to perform this step correctly will cause the server to terminate whenever the administrator logs off from the server machine.

Managing User Profiles

Each application has a runtime INI file in which its various settings are stored. The JIS Server allows for a separate runtime INI file to be assigned to each user that connects to it. This INI file contains the user's personalized settings. In runtime, the settings contained in the personalized INI file override the corresponding settings in the application's INI file.

There are two ways to access the INI file containing the user's personal settings:

- By creating a separate HTML file for each user
- By specifying the user's name through the LoginLauncher

The User's INI Files Location

When installing a JIS for Java runtime application for an end-user who requires a personalized application setting, a special user's runtime INI file should be created. Create a separate INI file for each application the user may run. The user's runtime INI files are kept on the server computer under a separate directory.

The INI file for the user's <username> for application <applname> should reside under the runtime root directory:

`$RuntimeRootDir\users\<username>\<applname>.ini.`

Where `$RuntimeRootDir` represents:

`<Product installation folder>\JacadaFiles` - in the product installation.

`<Runtime installation folder>` - in a runtime installation.

Example

For **User=Brad** working on **Application=Support**, assuming the JIS for Java runtime installation root directory is c:\JISRuntime, create the following file:

`c:\JISRuntime\users\brad\support.ini`

Creating a separate HTML file for each user

A User can be associated with the personalized INI file through an html file specially created for this purpose. The user's URL address points to the specific html file which in turn calls the personalized INI file. To make this feature work

- Create an html file for each user. You do so by duplicating the original application html file and renaming it.
- In the user's html file, add the "Profile" parameter with the user's name.

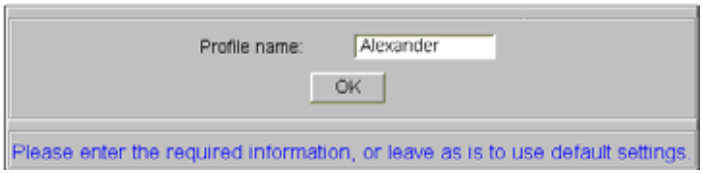
Example

```
<PARAM name = "Profile" value = "TEST">
```

Use this option when you have a small number of users. A possible scenario is when two types of users work on an application in different modes: some in test mode and some in live mode. Two separate html files will enable each user to directly log into the desired mode.

Using the LoginLauncher

The LoginLauncher enables users to access their personalized ini file by simply typing their name in the launcher's edit box. To activate this feature, in the application's html page change the RequestProfile parameter default value to **true**. As a result, when the client connects to the server, the following screen is presented in the browser's window:



If you have set the **Profile** parameter with the user's name, this name will be displayed in the **Profile name** field, as seen in the example above. If not, you can type the user's name in the empty **Profile name** field. The server looks for the user's personalized INI file. Should such a file not be found, an error message will be issued.

If you choose to use the default INI setting, leave the Profile name field blank. In either case, clicking the **OK** button establishes a connection with the server.

Use this option when you do not want to create a separate html file for each user, e.g. when a large number of users requires access to separate INI files.

Maintaining the User's Application INI File

To add a user to an application

- For the first user you define, create a **users** directory under the `$RuntimeRootDir` directory explained above.
- For this user's first application, create a <username> directory under `$RuntimeRootDir\users`.
- Create an <applname>.ini file containing the user's specific parameters and place it in the user's directory. Note that this INI file should only contain the user's specific parameters.

Remember: In runtime, the personal settings in the User's ini file override the corresponding default application ini file settings.

To remove a user from an application or all applications

- Remove a user's access to a specific application by deleting the `$RuntimeRootDir\users\<username>\<applname>.ini` file.
- Remove a user from all the applications by deleting the user's directory including all the files in it.

Note: When working in debug mode, a message is sent to the Client's debug log stating whether the user profile INI file was found or not.

If the INI file is found, the message reads - "User-profile ini found".

If the INI file is not found, the server sends the path where the INI file was looked for.

Chapter 4. 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 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.

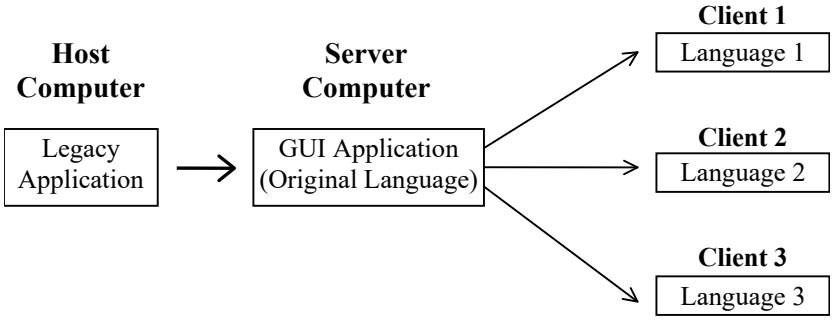


Figure 22. Language localization schema

This chapter describes:

- How the Localization Feature Works
- The Resource Files
- Setting the Runtime Localization Mechanism
- String Types Handled by Localization
- Debugging your Localized Application
- ISO Language and Country Codes
- Current Limitations

How the Localization Feature Works

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.

Localization Feature Workflow

The localization feature workflow is:

1. During the compilation process, a resource file is created and placed in `<InstallDir>\JacadaFiles\classes\appls\<ApplName>`
   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:
5. 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.
6. The application runs displaying the translated strings.

Activating the Localization Feature

Note: If the web server on your JIS Server machine is Internet Information Services (IIS) 6.0, `".res"` must be defined to IIS as a valid MIME type. (IIS 6.0 is the web server that is provided by Microsoft as part of Windows 2003.) If `".res"` is not defined to IIS 6.0 as a valid MIME type the localization feature will not work, because the client code will fail to load the resource files.

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 three resource files:

- An Original resource file which is created during compilation
- A Translated resource file which should be created for each additional language
- A General resource file that includes strings common to all Java Client runtime applications.

The Original Resource File

When you generate a runtime, a text file named `StringResource.res` is generated and is placed in the following directory:

```
<InstallDir>\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: "`<OriginalString>`" [`<space>`] =

Example 24. Original resource file

```
"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_<LocaleCode>.res`

Note: You must use the country and language code exactly as they appear in the Java standard Locale code.

2. Append the translated strings to the original language strings in the following manner:

   `"<key>"[<space>]=[<space>]"<value>"<line break>`

Example 25. Translated resource file

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.

The General Resource File

JIS Interface Server provides a number of predefined string resources in several languages. These application-independent strings form part of the Java Client. You should not modify these strings. However, you can add support for additional languages. For this purpose a file containing only the original application key-strings is provided. The `StringResource.res` file is placed in the following directory:

`<JavaRootDir>\JacadaFiles\classes\cst\client\resources.`

The work procedure for creating a General source file for an additional language is identical to that of creating a Translated file based on an Original resource file.

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 language the client displays is defined via the following three mechanisms. Note that when two or three mechanisms are implemented simultaneously, the one with the highest priority takes precedence.

- Setting through the system default language of the client Java VM. This is the default choice and has the lowest priority.
- Setting through the application HTML file. This has the second priority.
- Setting through the JacadaStarter API. This has the highest priority.

Note: When localization is disabled, the runtime always runs using the original host language.

Setting Through the Application HTML File

The language that the client displays can be defined through the Language parameter setting in the application HTML file. When the client places a request for an applet, the `Language` parameter points to the `StringResource_<LocaleCode>.res` file that is associated with it.

The (optional) parameters are:

```
Locale_Language          = <LanguageCode>
Locale_Country = <CountryCode>
Locale_Variant = <Variant>
```

Set these parameters to define the language variation that will be loaded to the client. The third parameter allows you to choose a regional variant. Note that the value for the variant parameter is vendor and browser-specific.

Example 26. Setting through the application HTML file

```
<parameter  name="Locale_Language" value= "fr">
<parameter  name="Locale_Country" value= "CA">
```

The parameters set in this manner point to the resource file containing strings translated into Canadian French.

In addition, for a multi-lingual clientele, you can create a single HTML file featuring an index of languages. Clicking a language name links the client to the HTML file containing a call to the relevant translated resource file.

> **Note:** The Locale_Language parameter is generated during compilation when the Localization feature is enabled. If not specified otherwise, the default value is English.

Setting Through the JacadaStarter API

You can control the language localization setting by extending the JacadaStarter class. Use this class to predetermine which translated resource file is downloaded to the client during the initialization process or create an interface to allow the user to choose from among several languages.

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
- Menu options
- Main Window (application independent strings)
- Subapplication
- Bubble help
- Labels
- Tab controls
- Button / check box / option box labels
- Error messages
- HTML—from client, server
- Message Box—from host, client, server
- Message box buttons (Yes, No, OK, Cancel, etc.)
- About dialog box text and buttons
- Date: different formatting possibilities, Day and Month display on the UI.
- Numeric fields: different formatting are supported.

Example 27. String types handled by localization

The number "1,000,000.99" will appear as "1 000 000,99" under a French Locale. To set a specific language Locale you must change the default setting in the application HTML file to the required language code. When typing the number, the user will have to type according to the definitions of the Locale.

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 visa versa will also result in changes in the newly generated resource file. Static strings are written to the resource file, dynamic ones are not.

Debugging your Localized Application

The localization feature includes a mechanism for testing the translated application for eventual errors. You may wish to do so on two occasions:

- When a problem arises and you want to trace its source.
- When you want to QA the results of applying the Localization feature, before transmitting the final product to the end user.

To test your translated application you will have to work in debug mode. In the following sections you will learn how to activate this mode and use its tools.

How to Work in Debug Mode

To find problems and errors, work in debug mode. The application runs in debug mode when the LocaleDebugMode parameter in the application html file is set to `TRUE`.

Example 28. How to work in debug mode

```
<PARAM name = "LocaleDebugMode"   value = "TRUE">
```

As a result of activating debug mode, two things happen:

1. In runtime, question marks (?) and asterisks (*) indicate that an error has occurred.
2. A section detailing all the errors found while executing the Localization feature is added to the browser's log file.

On the Runtime Window

Question marks and asterisks on the runtime window indicate that an error has occurred. You have to go through each and every window to verify that there are no question marks or asterisk signs. The lack of such signs indicates that the translated application is error free. These two signs indicate the following:

Question Mark

Indicates that a key is missing in the resource file. The question mark is followed by the original string.

Example 29. Question Mark

```
?MyMissing OriginalString
```

The key can be missing for several reasons:

- The resource file was not found
- The line in which the specific key was written was not found
- A syntax error in the key prevented it from showing

Asterisk

Indicates that a value (a translated string) is missing in the resource file. The asterisk is followed by the original string.

Example 30. Asterisk

```
*MyOriginalString
```

In addition, the asterisk indicates which original strings will be translated during the Localization process. Consequently, any string without an asterisk preceding it is by default dynamic.

The Log File

The existence of asterisks or question marks is an indication of a problem. To find out the nature of the problem, refer to the log file. In the Localization section of the Browser's log file you get a "live" display of the translated resource file as it is loaded into the cache.

In the log file find out:

- Whether the Translated Resource file was actually found.

  When a file is missing, a MissingResourceException message is written to the log.

  A file can be missing for several reasons. For example, it could be placed in the wrong directory, or the file name could be wrong.
- The parsing status.

  This includes a list of valid pairs of original and translated strings, missing values, syntax errors, and comments. The list shows the line number in the resource file and the type of error found in that line.

- Which original string (keys) are requested and which translated strings are retrieved during the creation of each new window.

  The translated version of a string does not appear in the window when:
  - The string was defined as dynamic rather than static.

    In this case no request for its translation will be issued
  - The string in the resource file has been commented out.
  - The translation for the original string has not been provided.

Note: The log file gives a detailed report of syntax errors and missing translated strings. However, keys missing for whatever reason are not indicated.

How to enter the log file:

- For an application running under Microsoft Internet Explorer, view the log file in the following directory: `<WindowsInstallDir>\java\javalog.txt`

  To enable the logging option, in the browser choose Tools > Internet Options > Advanced, and under the Java VM heading, set the Java logging enabled check box.
- For an application running under Netscape (Navigator or Communicator), view the log file in the browser's Java Console.
  To open the Java Console:
  - In Netscape Navigator choose from the menu: Window > Java Console
  - In Netscape Communicator choose from the menu: Communicator > Java Console

ISO Language and Country Codes

The localization feature uses the two letter language code and country code standard. 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 32 constitutes an example of some locales.

Table 32. Locale examples (Sheet 1 of 2)

Language Name	Language Code	Country code
English (Australian)	en	AU
English (Canadian)	en	CA
English (United States)	en	US

Table 32. Locale examples (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 string and the translated one 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.

Chapter 5. Printing Features

This chapter describes the following printing features:

- Host-to-Client Printing - Saving host print jobs on the server or sending them to the client to be printed from the local printer

  Use the printing emulation feature to print Host print jobs from a printer connected to your desktop computer, or to save print jobs on the JIS Server. The printer emulation includes an API that opens up the printing feature to a wide range of possibilities.

  The printing emulation feature operates on both the client and the server. In order to be operational however, the printing feature must be configured. In this chapter you will learn how to implement the printing emulation feature. Note that this feature supports all SBCS languages supported in the display emulation.
- Extending the Printer Emulation

  Use the GUI printing feature to print windows currently present on your screen. The information that will be printed depends on the way the windows are configured to display in runtime.
- Printing the Client Host Screen

  Use this feature to print the Host screen currently displayed on your client.

Host-to-Client Printing

This section deals with the issue of host-to-client printing.

The Host-to-Client Printing Architectures

- The printer emulation connection is initiated through the JIS Server. Data received from the host is saved on the server or relayed to the client for printing.

  The main advantage of using the JIS Server as a means of connection is that the JIS Server allows the printer emulation to connect to the host also when the host is protected by a firewall.

Host to Client Printer Emulation Connection via JIS Server

The figure below and the explanation that follows describe the activity that takes place from the moment a client connects to the JIS Server to the point that the host sends print jobs to the printer connected to the client.

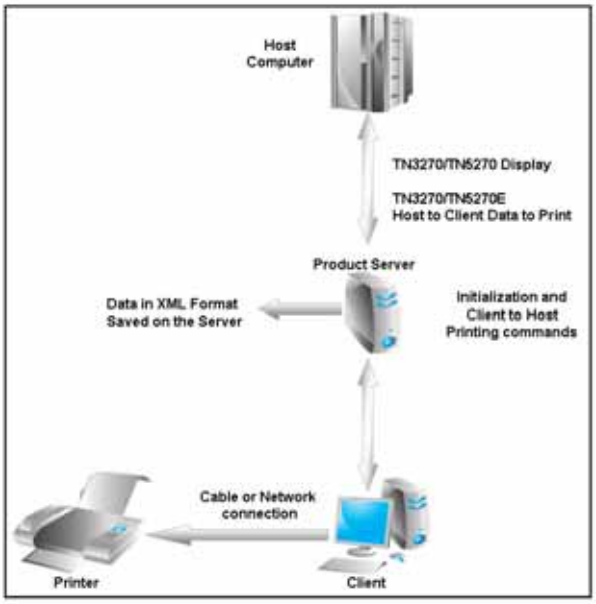


Figure 23. Host-to-client printer emulation connection via JIS Server

1. The client connects to the JIS Server by loading an HTML page. Application initialization classes are sent back to the client.
2. The client runs the Application. As a result:
   - The JIS Server initiates a connection with the host computer.
   - Printer emulation classes are sent to the client. The client creates a printer emulation session.
3. The client connects to the host using TN3270E (for mainframe) or TN5250E (for iSeries) via the connection established by the JIS Server.
4. The Client sends printing requests, via the JIS Server, to the Host application.
5. The host returns printing jobs, via the JIS Server. The print job is saved on the server for further use or sent to the client for printing.

Making the Printer Emulation Operational

Before you can use the printer emulation, you must perform some preliminary operations.

To implement this feature:

- Configure the host to recognize the Printer's Logical Unit (LU) name.
- In the `<ApplName>.ini` file:.
  - Set printing parameters.
- In the application HTML file, add a parameter that reads the printer emulation archive.
- If you wish to enhance the printing capabilities, extend the printing emulation using the printing emulation API.
- Order the initialization of a printer emulation session. This is achieved either through an INI file flag, or by activating an JIS Interface Server DoMethod. See Chapter 6 - "Extending the Java Code" on page 193.
- Make sure you have printing authorization on both the Client and the Server side.
- In the runtime environment, run the application.

Configuring the Host to Recognize the Printer LU Name

In order to configure the host to recognize the printer LU name, follow the instructions in this section.

Mainframe

The connection between the client and the mainframe is carried out using TN3270E. Specification of the printing parameters is done in the runtime `<ApplName>.ini` file.

iSeries

The iSeries must support TN5250E.

In addition, The iSeries system administrator will have to configure the Server to recognize the printer LU.

Establishing Print Parameters in the <ApplName>.ini File

All parameters concerning connectivity between the client and the host, via the JIS Server, as well as parameters defining print settings, must reside in the `<ApplName>.ini` file. The majority of these parameters are added manually. The `<ApplName>.ini` file resides under the following directory:

```
<RuntimeInstallDir>\appls\<ApplName>\RT32\<ApplName>.ini
```

Following is the list of parameters that must reside in the `<ApplName>.ini` file

[GUISys TN3270] or [GUISys TN5250]

`Printer`	Set the value to 1 to open an emulation session on the client. This parameter is automatically written to the `<ApplName>.ini` file during the compilation process.

[TN3270 Printer] or [TN5250 Printer]

`Host`	Name of host with which you wish to establish a connection. When this value is left empty, it is assumed that the host is the same one as for the display session.
`Port`	The number of the communication port. When no value is given, the default is the port number used for the display session.
`LUName`	The LU name for the printer. When this is not specified, the default printer is the one associated with the current display session. **Note**: When printing from an iSeries, this setting MUST be specified, since TN5250E does not automatically associate a printer with a display session.
`DefaultRows`	Number of text lines in a page. The default is 66.
`TimeOut`	Number of seconds the printer emulation waits before sending the text to the printer. For any additional text that arrives during this period, the count will start over.
`DefaultColumns`	Number of characters in a line. The default is 80. The host may specify a number that will override this value.

`IgnoreHostWidth`	When set to a non-zero value, the printer emulation will ignore any attempt by the host to specify a page width and will use the value specified in the DefaultColumns setting. Note: When printing from an iSeries, this setting is ignored.
`WrapText`	When set to a non-zero value, the printer emulation will wrap any line that is longer than the current line width, onto the next line.
`MarginUnits`	Defines the units of measure used to specify the margin values: 0 = inches, 1= millimeters.
`LeftMargin`	The minimum distance from the left edge of the page to the point where a line begins.
`RightMargin`	The minimum distance from the right edge of the page to the point where a line ends.
`TopMargin`	The minimum distance from the top of the page to the first line.
`BottomMargin`	The minimum distance from the bottom of the page to the last line.
`IgnoreFFAtFirst`	Suppress form feed when it comes first, before any printed page. Set to 1 to enable. The default value is 0.

Adding HTML Parameter to Read Printer Emulation Archive

In order to use the printer emulation feature, printer emulation classes and settings must be downloaded to the client. These classes are contained in the following files:

`clbase & clprint.jar`	For users of Sun Java plugin

`clfull-signed.jar`	For users of Sun Java plugin, requiring signed applets

These archive files are not automatically loaded to the client from the Web Server during the initialization of an application since they are not referenced in the default HTML settings. If you wish to enhance your application with printer emulation capabilities, in the application html file change the default reference to the basic archive clbase.jar with one of the archives mentioned above.

Initializing a Default Printer Emulation Session

There are two ways you can order the initialization of the printer emulation feature:

- Through an INI file flag
- By activating a special JIS Interface Server DoMethod

Initializing the printer emulation through the INI file

In the `<ApplName>.ini` file, set the following parameter:

```
[GUISys TN3270] or [GUISys TN5250] or [Emulator]
Printer=1
```

Initializing the printer emulation by activating an JIS Interface Server method

You can order the initialization of the printer emulation through a call to the following DoMethod:

```
OpenPrinterEmulation
```

Note that the method must be triggered by an action defined by the developer.

In addition, the following two DoMethods control the behavior of the printer emulator:

`ClosePrinterEmulation`	Closes an open printer emulation session.
`PrinterEmulationLUName`	Returns the last LU Name under which a printer emulation session was opened.

Tracing Printer Emulation Problems

Use debug mode to trace problems in your printer emulation feature. The information is written to a file you create for this purpose.

To run the printer emulation feature in debug mode, configure the `<ApplName>.ini` file as follows:

[TN3270 Printer] or [TN5250 Printer]

`TraceLevel`	Determines the amount of information that will be logged to the trace file.
`TraceFile`	Determines the name of the file to which the trace information is written. The default file name is tn_print.trc.
`ResetTraceFile`	Determines whether or not to reset the trace file before a new session is created. The default value 1 resets the trace file.

Using the Printer

The printer emulation feature is ready for use as soon as printer emulation settings are defined. The end user need only run the application.

An error message appears when the printer is unable to connect.

Print Options Dialog Box

Each time the host sends a print job to the client, the client displays a dialog box. This dialog box is used to assign the current print jobs to a local printer, to determine the number of copies required, and to set other print related parameters.

Note: The dialog box is platform dependent and may look different under different platforms. The picture below is of a MSIE Print dialog box run on Windows.

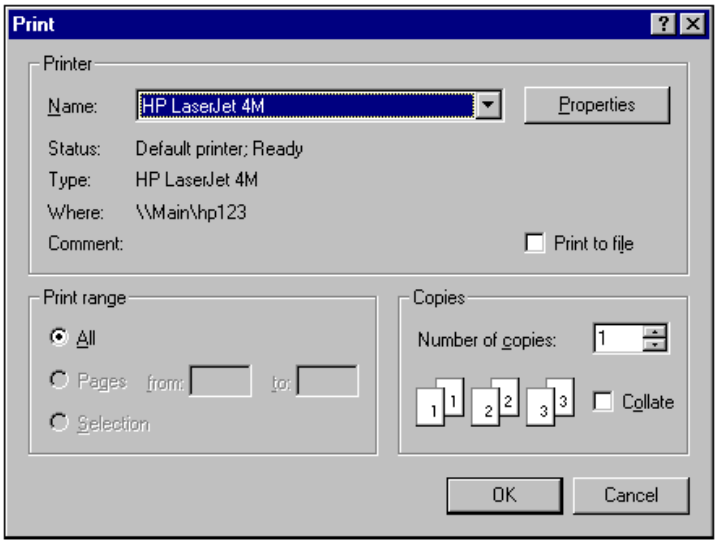


Figure 24. MSIE Print dialog box running on Windows

In addition the end user is provided with a printer emulation user interface. See “The Printer Emulation User Interface” on page 174.

Printing Via the Java Page Setup Dialog Box

In addition to using the printing features available in the operating system’s Print dialog box, you can also use the enhanced features available in the Java Page Setup dialog box. This feature is available using Java plug-in and allows you to further configure your page setup.

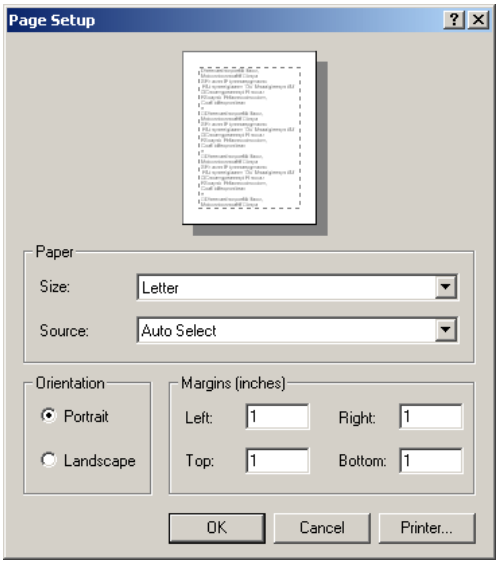


Figure 25. Java Page setup dialog box

Options Available in the Java Page Setup Dialog Box

The Java Page Setup dialog box allows you to set the following:

- Paper size.
- Paper source.
- Page orientation - portrait or landscape.
- Page margins.

You only need to set the options in the Java Page Setup dialog box once. The next time that you invoke the Java Page Setup dialog box, it opens with your previously configured settings.

Enabling and Disabling the Java Page Setup Dialog Box

You enable printing via the Java Page Setup dialog box by adding a setting to the `<ApplName>.html` file.

To enable the Java Page Setup dialog box:

In the `<ApplName>.html` file, add the following setting:

```
<PARAM name = "UseJavaPrintDialog" value = "true">
```

With Java plug-in 1.4 and higher, the Java Page Setup dialog box opens by default and you do not need to add this setting to the `<ApplName>.html` file. However, you can disable this feature.

To disable the Java Print dialog box:

In the `<ApplName>.html` file, add the following setting:

```
<PARAM name = "UseJavaPrintDialog" value = "false">
```

When set to `false`, the operating system's Print dialog box is used, even if you are working with the Java plug-in.

Setting Page Orientation

When using a Java plug-in you can set the page orientation to either Portrait or Landscape in the `<ApplName>.html` file. This setting is then the default for all printing jobs, including:

- GUI printing
- Host View printing
- Printer emulation

To set the default page orientation:

In the `<ApplName>.html` file, add the following parameter:

```
<PARAM name = "PrintPageOrientation" value = "Landscape">
```

OR

```
<PARAM name = "PrintPageOrientation" value = "Portrait">
```

The Printer Emulation User Interface

Use the printer emulation user interface for two main purposes:

- Checking and manipulating the printer status
- Changing printing parameters

To enter the user interface

1. Run the application
2. From the Runtime menu choose Application > Printer Emulation. The Printer Emulation dialog box opens.

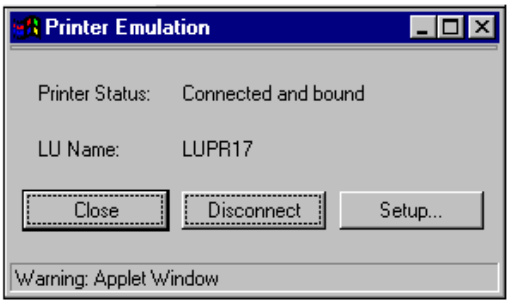


Figure 26. Printer emulation user interface

> **Note:** The printer emulation feature must be defined as enabled in the `<ApplName>.ini` file.

The Printer Emulation Dialog Box

In this dialog box find the following information about the printer:

`Printer Status`	The state of the printer. Possible values are:
	`Disconnected`
	`Connecting`

`Connected and not bound`	The host recognizes the printer but no program is using it.
`Connected and bound`	The printer is being used by a program.
`LU Name`	The LU name by which this printer is known to the host or gateway.

In addition, the dialog box displays the following buttons:

`Disconnect /Connect`	This button's text alternates between **Disconnect** and **Connect** depending on the printer emulation's current state. The button is disabled while the printer is connecting and while it is servicing a print job.
`Setup`	Opens the **Printer Emulator Setup** dialog box.

The Printer Emulation Setup Dialog Box

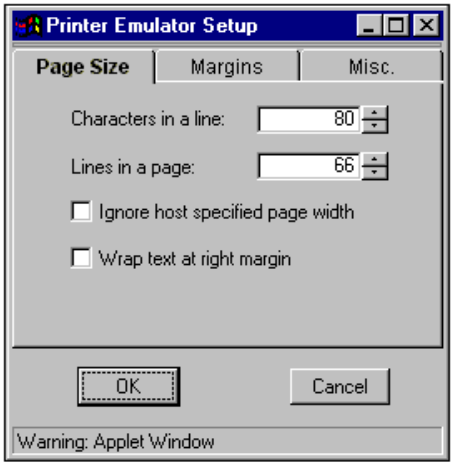


Figure 27. Printer Emulation Setup dialog box

This dialog box contains three tabs. Table 33 lists these tabs, the settings they contain and their corresponding `<ApplName>.ini` parameter names:

Table 33. Settings in the Printer Emulation Setup dialog box

Tab	Settings	INI Parameter Name
Page Size	Characters in a line	`DefaultColumns`
	Lines in a Page	`DefaultRows`
	Ignore host specified width	`IgnoreHostWidth`
	Wrap text at right margin	`WrapText`
Margins	Measurement units	`MarginUnits`
	Left margin	`LeftMargin`
	Right margin	`RightMargin`
	Top margin	`TopMargin`
	Bottom margin	`BottomMargin`
Miscellaneous	Time out	`TimeOut`
	Ignore leading form feed	`IgnoreFFAtFirst`

The dialog's OK button is disabled while the printer emulator is servicing a print job. The button will also be disabled if any parameter entered is illegal.

Limitations

The limitations of this feature is:

- Color printing is not supported.

Troubleshooting

The printer may fail to print for various reasons. When this happens, the system issues an error message describing the nature of the error, for example:

Figure 28. Error message on printer fail

Printing could fail for various reasons:

Table 34. Printer fail reasons and solutions

Problem	Solution
The server does not exist.	
The Server does not support TN3270E/TN5250E.	Where possible, install the TN3270E/TN5250E emulator. Otherwise, connect to a gateway that does support TN3270E/TN5250E.
The Server does not support printing.	Define the printers logical name on the Server and on the Host.
The LU name is invalid or belongs to a display terminal.	In the `<ApplName>.ini` file, write the correct LU name.

Extending the Printer Emulation

The printer emulation API opens up the printing feature to a wide range of possibilities.

Enhancements can be made in three major fields of the printing domain; you can decide upon:

- The destination of the printing data. This can include sending the printing data to the client, saving the data on the server, sending a URL to the client, etc.
- The printing format: disk files, template reports, html forms
- The printing style: Font size, Graphics, printing from right to left, etc.

The printing emulation feature enables the creation of printer classes both on the Server side and on the Client side.

The Printer Emulation API Architecture

Below is an illustration of the optional ways to use the printer emulation:

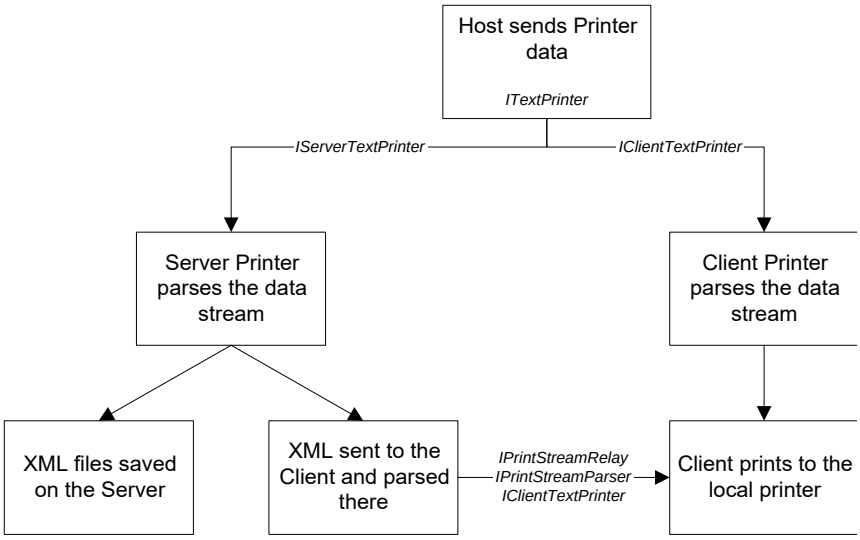


Figure 29. Optional ways to use the printer emulation

Data sent from the host can be put to use in several ways:

- Printer emulation contents saved on the Server. The IServerTextPrinter interface parses the printer data stream, as received from the host, saves the print job in XML form, and merges it into a report, prints it using some system script, etc.
- Depracated direct Client printer emulation. The client receives the raw printer data stream, directly from the host.
- Client printer emulation through the printer proxy. The printer mechanism is split between the Server and the Client. On the Server side an interface parses the printer data stream, as received from the host, and sends the contents of the page, in XML form, to the client. On the client, a proxy receives the contents, and sends it immediately to the final printer class. This is done using the IPrintStreamRelay, IPrintStreamParser and IClientTextPrinter interfaces.

Creating a Printer Emulation Instance

To initiate a printer emulation instance, the appropriate instance name and the classes implementing it must be registered and identified.

Determining the Printer Emulation Type

The printer emulation type is determined in the [Printing Handlers] section of the application runtime.ini file. For each different type of printer emulation, there is a handler name and two class names

`Handler name`	The handler identifies the printer emulation type.
`Class name`	The classes that implement the Printer Emulation API. The first class name relates to the Server side, the second to the Client side.

The parameter containing the handler and classes looks as follows:

```
[HandlerName]=[ServerClass],[ClientClass]
```

Example 31. Parameter containing the handler and classes

```
[Printing Handlers]
Printer1=HostToServer.class,ClientToPrinter
```

The Default Settings

The default handler name is ‘Default’. The class names may contain a user extension, or use the JIS provided defaults. Two keywords are defined: ‘_none’ and ‘_default’. The first indicates that no class should be used, the second – use our default. Otherwise, the name is treated as a class name. In the following table are listed some simple examples for usage of the new parameter.

Table 35. Default handler settings

Parameter Setting	Server	Client
Default= _default,_default	Data sent to the Server *ServerTextPrinterToXML*	Data relayed to the Client *IPrintStreamRelay*
Default= _default,_none	Data sent to the Server *ServerTextPrinterToXML*	No class is created

The Extension Class Path

The server reads the class names from the runtime.ini file, according to the handler name. The class names are passed to the client, so that both server and client may load extension classes if requested. The classes are searched for in a fixed path.

Table 36. Extension class paths and locations

Location	Classpath
Server	`cst\server\printing\<ClassName>.class`
Client	`cst\client\printing\<ClassName>.class`

Initializing an Extended Printer Emulation Session

To initiate an extended printer emulation session or to allow activating a printer by its handler name, call the following DoMethod:

1 OpenPrinterByHandler

This DoMethod has two parameters:

`HandlerName`	Enter the printer's handler name. This name must correspond to the name that appears in the Printing Handlers section of the runtime.ini file. The name must be enclosed within quotation marks.
`LUName`	Enter the LUName the new printer requests from the host. The name must be enclosed within quotation marks.

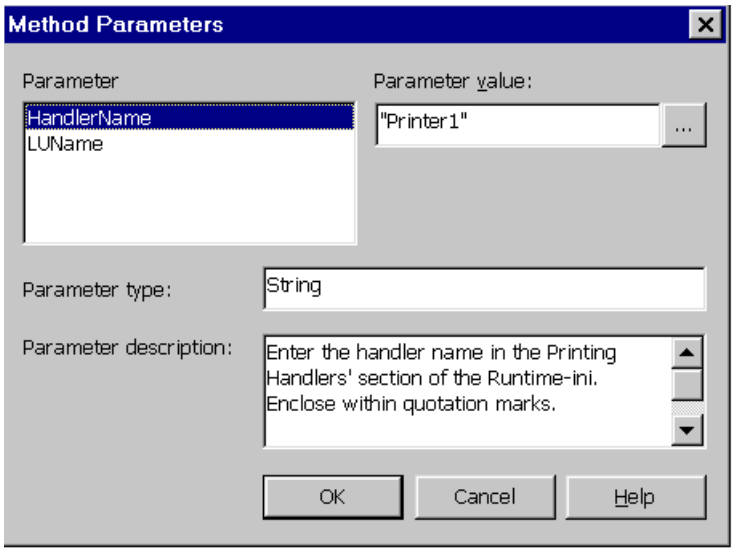


Figure 30. Method Parameters dialog box

The Application Runtime.ini File Settings

This section describes the application runtime.ini file settings.

General runtime.ini file Parameters

The following parameters must appear under the [TN3270 Printer] or [TN5250 Printer] section of the application runtime.ini file

Table 37. Parameters in [TN3270 Printer] or [TN5250 Printer] (Sheet 1 of 2)

Parameter	Description
`WorkRootDirectory`	The location on the server where all the XML files are saved. This parameter is obligatory when using the server printing option. **For Example:** `c:\java\xmls\user_1_1\fred_1_1.html`

Table 37. Parameters in [TN3270 Printer] or [TN5250 Printer] (Sheet 2 of 2)

Parameter	Description

`SpoolDirectory`	Prefix for the session print jobs folder. This parameter is obligatory when using the server printing option.
	For Example:
	`SpoolDirectory=user`
	`c:\java\xmls\user_1_1\fred_1_1.html`

`BaseFileName`	File name for the XML pages. The name is appended with a unique ID.
	For Example:
	`BaseFileName=fred`
	`c:\java\xmls\user_1_1\fred_1_1.html`

`PagesToBuffer`	Maximum number of XML pages to buffer on the server, before sending them to the client.
`UserDefinedParameters`	A string containing user parameters used by new extensions.
`CompressStream`	Determines whether to instruct the server to compress the XML pages before they are sent to the client. The recommended setting for this parameter is the default 1.

Special Runtime.ini Parameters Used with JIS Examples

Table 38. Special runtime.ini file parameters

Parameter	Description
`WorkRootURL`	URL address, mapped to the server's WorkRootDirectory, where HTMLs are saved. This is sent to the client. **For Example:** `http://localhost/xmls/`
`XSLTforXMLtoHTML`	URL address for the XSL file that is used by the HTML extension. **For Example:** `file:///c:\java\src\cst\server\printing\XMLtoHTML.xsl`

HTML Parameters

When the client classes are created, the IPrintStreamPrinter class is created by using the runtime.ini parameter for the client class name. This class receives an IClientRuntimeUtilities implementation, which it uses to query in the HTML file for the IPrintStreamParser class. This second class also receives the same implementation, looks up the HTML file for the IClientTextPrinter class, and creates it. To create these two classes, the following parameters were added to the HTML file

Table 39. HTML parameters

Parameter	Meaning	Default Class Name
`PrintStreamParser`	Implements IPrintStreamParser	`XMLPrintStreamParser`
`ClientTextPrinter`	Implements IClientTextPrinter	`ClientTextPrinter`

Sending the Print Stream to the Client

The server implementation by default sends the parsed printer data to the client. The printer data is sent in an XML format.

The XML format is used to refashion the host printer data stream, rendering it easier to handle and convert to other formats. The JIS implementation creates the XML on the server, using ServerTextPrinterToXML implementation of ServerTextPrinter. The XML may be sent to the client, or saved on the server.

The XML tags we use are defined in the interface IXMLPrintConstants. Extensions may use XML technologies such as XSLT that translates XML into other formats, such as PDF, RTF, or HTML.

Providing Client and Server Security Permissions

This section discusses client and server security permissions.

Client Security Permissions

Printing on the client demands security permissions; extensions written for the client must be granted permissions for any type of access. Extensions may also require signing the applet.

Server Security Permissions

The JIS printer classes on the server have Java application's permissions. This allows files and the printer to access the server. However, to allow file system access, so different session's print jobs can be separated and secured, an identification mechanism has been introduced.

Every extension is provided with a unique identifier, per session. The identifier consists of both the session and the process number on the server. This, combined with a time stamp provides a unique ID per user per print job. The identifier is used for creating a folder for each user, and putting in it all the print jobs. Each folder is deleted when the session closes. This is done through a call to a server-side interface method; the method removes the storage object upon termination of the session.

The path to the folder where all the users' sub-folders are to be created must be defined in the WorkRootDirectory parameter of the runtime.ini file.

Examples of How to Use the Extended Printer Emulation

This section provides examples of how to use the extended printer emulation feature.

Saving Data on Server Example

This example demonstrates how data sent from the host is saved on a specific directory on the Server for later use.

In this example, the following activity takes place:

- A server side class is added.
- The server class receives the data stream from the host.
- The server saves the data sent from the host on a designated directory.

Application runtime.ini Configurations

`Default=_default,_none`	Server side class is added.
`WorkRootDirectory=c:\xmls`	The root directory for the location of the XMLs on the server.
`SpoolDirectory=job`	The specific location of the XMLs on the server, i.e. `c:\xmls\job\`.

HTML Printing Example

This example demonstrates how the Client is sent a URL address to the location of HTMLs sitting on the Server.

In this example, the following activity takes place:

- Two classes, on both client and server, are added. On the server side, the class implements IServerTextPrinter; on the Client side, the class implements IPageStreamRelay.
- The server class receives the data stream from the host.
- The server class translates the data stream into XML, then translates the XML into HTML using XSL file.
- The server class saves the HTML on the server file system.
- The client receives a URL to the file system location.

- A new browser session is opened, allowing to print the HTML from it.
- Files are removed when the session closes.

Application runtime.ini Configurations

`Default=ServerTextPritnerToHTML, URLRelay`	Server side and Client side classes are added.
`WorkRootURL=http://localhost/xmls/`	URL of the server file system. This is sent to the Client.
`WorkRootDirectory=i:\java\xmls`	The location of the HTML files on the server.
`SpoolDirectory=user`	Prefix for the folder name.
`XSLTforXMLtoHTML=file:///i:\java\src\cst\server\printing\XMLtoHTML.xsl`	File URL to the XSL.

Note: The HTML printing quality relies on HTML capabilities, and therefore **does not support** over-typing, margin sizes or data stream commands for changing the position of the output on a single page. Margin sizes can be set from the system printer dialog, opened by the browser. The classes in this example do not need to be signed.

Printing via the Server and the Client

This example demonstrates how to print on the client, using the standard DOS Comm Ports (LPT1, etc.). In this example, the following activity takes place:

- The main class receives data from the host and relays it to the client.
- A Third level client side class generates a string of text, then sends it to a Comm port using a utility class.
- The name of the Comm port is read from the application's HTML file.

Application runtime.ini Configurations

`Default=_default,_default`	Server side and client side classes are added.
HTML Configurations	
`ClientTextPrinter=ClientTextPrinterToSerial`	Third level, client side class is added.
`TextPrinterDestination=`	The Comm Port to use.

Note: The Comm printing feature is Windows-specific. It uses Java exec command, to refer to the OS shell, and print to the port. Printing using Overtype Bold and Underscore is **not supported**.

Printing the Client Window

Use the GUI printing feature to print windows currently present on your screen.

Each Subapplication is printed on a separate page. In addition, a window caption is printed above each Subapplication stating its title and the date and time it was printed. Note that Date and Time are localized. They are calculated according to the client's time zone.

Note: The applet must be granted permission to use the printing option.

Activating the GUI Printing Feature

The GUI printing feature can be activated:

- Through the runtime Application menu
- Through code extension

To activate the GUI printing feature through the runtime Application menu:

1 From the menu choose Application > File > Print GUI The Print dialog opens.

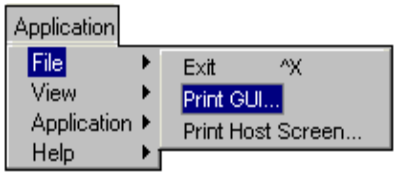


Figure 31. Print dialog box

2 In the Print dialog, press OK. The GUI will be printed.

To activate the GUI printing feature through Code extension:

Call the JacadaStarter's printGui() method.

Example32. Activate the GUI printing feature through code extension

In any Subapplication you can write:

```
getStarter().printGui();
```

Note: Adding a TRUE parameter–printGui(true)–causes the calling thread to block until the printing is done.

Eliminating the Print Setup Dialog

Each time you go to print a GUI screen, the Java client by default shows a print setup dialog, where you can modify the paper size and orientation and the page margins to be used. You can eliminate the display of the print setup dialog if you like.

To eliminate the display of the print setup dialog, add the following parameter to the HTML page:

```
<PARAM name = "ShowPrintDialog"    value = "false">
```

The default value of `ShowPrintDialog` is “true”. The parameter `ShowPrintDialog` works only when the Java client runs with a Sun Java plug-in.

Modifying the GUI Printing Feature Through Code Extension

You can enhance the printing feature by extending the code in the JacadaStarter class. The following methods can be overridden to meet your needs:

```
protected String getTitleToPrint (GUICSTPanel panel)
```

This method is called to determine which title to print above the Subapplication's panel. You can return any string you wish. The string can be separated into several lines using the newline ('\n') character. If null is returned, no title will be printed.

protected DateFormat getDateFormatToPrint ()

This method controls the format of the date and time in the printing. It should return a DateFormat object, which is a standard Java class (in package java.text). If the method returns null, the date and time will not appear in the printing at all. By default, the date and time are printed in the following format: "December 15, 1999 6:30 AM".

```
protected boolean needToPrint (GUICSTPanel panel)
```

This method is called to determine whether a Subapplication should be printed. By returning false, the user can prevent specific Subapplications from being printed.

Example 33. Preventing popup windows from being printed

The following code will prevent popup windows from being printed:

```
protected boolean needToPrint (GUICSTPanel panel) {
          if (panel.getPanelType() == GUICSTPanel.POPUP) {
               return false;
          }
          else {
               return super.needToPrint(panel);
          }
     }
```

Granting Permission to Print the Client Window

Due to security restrictions imposed by the Java Virtual Machine (JVM), the window printing feature can only work using signed files. JIS provides you with the clbase-signed.jar archive file to allow you to use the printing feature.

The HTML reference to the Client core CAB file:

```
<PARAM name ="cabbase" value="cst/clbase-signed.cab">
```

Note: The clfull-signed.jar archive file also allows you to use the printing feature but it contains other features you may not require.

If you try to run the application using an unsigned archive file, the following message will be displayed: "The application does not have permission to print".

If you run as an application and not as an applet, there are no security restrictions; the application can print by default.

Changing the Background Color of the Printed Window

The background color of a printed window can be changed. Typically, you will change the background color to white to optimize printing costs. Note that changing the Subapplication's background color affects also the background color of group boxes, frames, radio groups, tab folders, labels, check boxes and radio buttons.

You control the printed window's background color through the GUIPrintingBackground HTML parameter setting. The value set in this parameter must be an RGB color value in the standard HTML format. For example, if you wish to color the window's background white, set the "GUIPrintingBackground" parameter as follows:

```
<PARAM name = "GUIPrintingBackground"  value = "#FFFFFF">
```

Controlling the Scale of the Window's Printout

Screen resolution, printing settings and other printer's specifications may cause the window you wish to print to be too large to fit the printer's page.

You can control the scale of the printed window through the GUIPrintingScale HTML parameter setting. The value set in this parameter is a number representing the percent of the printout's full size.

Example 34. Controlling the scale of the printed window

```
<PARAM name = "GUIPrintingScale" value = "70">
```

will cause the printout to be 70% of its full size.

Note: Reducing the printout scale may cause some distortion and loss of detail in the printed output.

The following chart offers a gauge by which to choose the scale for the printed window. These approximated values are based on screen resolution, page orientation and paper size. The optimal value may depend upon the printer brand and the printing settings you selected.

Table 40. Choosing a scale for a printed window

Screen Resolution	Paper Size	Page Orientation	Recommended Scale
800x600	A4	Landscape	100%
800x600	A4	Portrait	70%
800x600	Letter	Landscape	96%
800x600	Letter	Portrait	73%
1024x768	A4	Landscape	77%
1024x768	A4	Portrait	55%
1024x768	Letter	Landscape	75%
1024x768	Letter	Portrait	57%

Printing the Client Host Screen

Use this feature to print the Host screen currently displayed on your client.

The Host printing feature can be activated:

- Through the runtime Application menu
- Through code extension

To activate the Host screen printing feature through the runtime Application menu:

1 From the menu choose Application > File > Print Host Screen. The Print dialog opens.

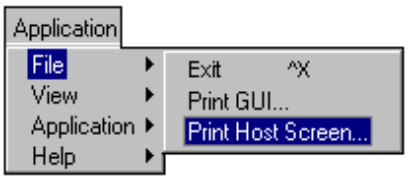


Figure 32. Print dialog

2 In the Print dialog, press OK. The Host screen will be printed.

To activate the Host screen printing feature using Code extension:
Call the JacadaStarter's printHostScreen() method.

Example35. Activating Host screen printing via code extension

In any Subapplication you can write:

```
getStarter().printHostScreen();
```

Note: Due to security restrictions imposed by the Java Virtual Machine (JVM), the Host screen printing feature can only work using signed files. For more information see “Granting Permission to Print the Client Window” on page 189. Adding a TRUE parameter–`printHostScreen(true)`–causes the calling thread to block until the printing is done. The printed copy presents a reverse image of the client host screen. Also, a window caption is printed above each host screen stating the date and time it was printed.

Chapter 6. Extending the Java Code

Java code is generated by JIS Interface Server for the Java client and the JIS Server. The Client Java code can be modified in various ways in order to add functionality beyond JIS Interface Server's capabilities, or change the appearance and behavior of the GUI components. The Server Java code can also be modified in order to solve security limitations and enhance performance. This chapter describes how to achieve this, and gives concrete examples which demonstrate the correct ways to do so.

This chapter describes:

- The Client Java Code Produced During Compilation
- Working with the Java Code

The Client Java Code Produced During Compilation

The following groups of files are generated during Compilation:

Java Sources in the Original Sub-directory

This code must not be edited by the user since it will be overwritten by subsequent executions of the Generate Runtime command. Files containing the Java source code are created as read-only.

- `<InstallDir>\JacadaFiles\src\appls\<ApplName>\original\Make.pl`

  A Perl script used to generate the Java sources and then compile them.
- `<InstallDir>\JacadaFiles\src\appls\<ApplName>\original\TempMake<#>.java`

  A temporary Java file used to store all the Java sources of a compiled application batch (those contained in the cst hierarchy, as well as in the user hierarchy).

Note: `<#>` stands for the batch number, e.g. `TempMake1.java`. The compilation process produces a `TempMake<#>.java` file for each compiled batch.

- `<InstallDir>\JacadaFiles\src\appls\<ApplName>\original\MainWindow.java`

  The main window is application-specific (and therefore generated during compilation) because it contains the tool bar which was defined in the application.
- `<InstallDir>\JacadaFiles\src\appls\<ApplName>\original\JacadaStarter.java`

  Contains the general initialization of the application.
- `<InstallDir>\JacadaFiles\src\appls\<ApplName>\original\<WindowName>.java`

  A Java code that generates the window. A java code is generated for each Subapplication window of the application.
- `<InstallDir>\JacadaFiles\src\appls\<ApplName>\original\ApplSubApplWindow.java`

  Contains the application wide Floating Menus.

Java Sources in the User Sub-directory

The user's sub-directory files is where the user is expected to write his own Java code. During compilation some skeletons are generated for the user to fill in. These files will not be overwritten when other occurrences of the compilation process take place. Therefore, any editing the user may have done will not get lost.

The following Java source files exist in the User subdirectory

- `<InstallDir>\JacadaFiles\src\appls\<ApplName>\user\ApplSubApplWindow.java`

  This class is a common parent of all the application's windows.
- `<InstallDir>\JacadaFiles\src\appls\<ApplName>\user\Applet.java`

  This class invoked from the HTML file. It activates the JacadaStarter class which launches the Java Client runtime application.
- `<InstallDir>\JacadaFiles\src\appls\<ApplName>\user\JacadaStarter.java`

  Contains the general initialization of the application, such as registration of the application's windows that the user wishes to extend. This class extends the JacadaStarter class which resides in the `<InstallDir>\JacadaFiles\src\appls\<ApplName>\original` directory.
- `<InstallDir>\JacadaFiles\src\appls\<ApplName>\user\ValidityCheck.java`

  Contains the Skeleton for extending the default validity checks of the Java Client. It is the user's responsibility to update the class for additional checks.

Compiling the User's Java Sources

Use the `<InstalDir>\JacadaFiles\src\appls\<ApplName>\user\jacc.bat` batch file to compile Java sources which you have modified.

These sources should reside in the directory:

`<InstallDir>\JacadaFiles\src\appls\<ApplName>\user`

Automatic Overwriting of User Files During Version Upgrading

During the compilation process, User files are created. These include the following Java files:

`<InstallDir>\JacadaFiles\src\appls\<ApplName>\user\ApplSubApplWindow.java`

`<InstallDir>\JacadaFiles\src\appls\<ApplName>\user\Applet.java`

`<InstallDir>\JacadaFiles\src\appls\<ApplName>\user\JacadaStarter.java`

`<InstallDir>\JacadaFiles\src\appls\<ApplName>\user\ValidityCheck.java`

and the following HTML files:

`<InstallDir>\JacadaFiles\<ApplName>.html`

`<InstallDir>\JacadaFiles\<ApplName>-signed.html`

`<InstallDir>\JacadaFiles\<ApplName>-start.html`

These User files will not be overwritten during subsequent compilation operations. However, when upgrading to a new version of JIS Interface Server, the application's User files will be automatically updated, if needed. Note that the changes the User has made to the file are not automatically updated.

Whenever a user file that was modified by the user is overwritten through the upgrading process, a backup of the original file is created in the same directory. This file bears the original file's name with an `*.old` extension.

After generating a runtime with a new version of JIS Interface Server for the first time, the Generate Runtime log window will report which User files have been updated. Edit the User files that have been updated, and insert your modification back into them, using the `*.old` backup files as reference.

After modifying a file in the user directory, run `jacc.bat` to recompile the Java classes.

When a new version of Applet.java is created in the user directory, JIS Interface Server updates the Applet.java file and prepares a backup file named Applet.old in the user directory. After the updating has taken place, the user should copy the changes from Applet.old back into Applet.java.

Java .class Files

After the Java sources have been generated, the Generate Runtime process invokes the Java compiler which generates Java `.class` files which reside for example, under the `c:\Ace\JacadaFiles\classes` directory. Any error in the compilation of the Java code aborts the Generate Runtime process.

JIS's Javadoc Files

JIS provides you with a set of html files that include explanations about how to work with the generated classes and the methods they contain.

The javadoc files are installed under the following directory:

```
<InstallDir>\JacadaFiles\docs\client
```

Note: Only public methods are documented in the javadoc.

Working with the Java Code

This section describes working with the Java code.

About Event Handling

JDK 1.1 defines the Delegation Event Model, which replaces the event model of the earlier JDK versions. JIS Interface Server uses the newer model. Since it is inadvisable to mix the two models in one application, any event handling that your extended code performs should be according to the Delegation Event Model.

About Deprecated Methods

Some methods defined in JDK 1.02 were replaced by different methods in JDK 1.1, and were marked as deprecated. When writing Java code, you should avoid using these deprecated methods, and choose the newer versions instead. This is especially true when extending JIS Interface Server Subapplications—JIS's components use only the newer versions of the methods, and using the deprecated methods might yield unexpected results. You can easily identify such methods by the "deprecated" indication in their API documentation. Usually, the newer version of the method will also be noted there.

For your convenience, here is a list of the most commonly used deprecated methods, and their replacements.

Table 41. Depreciated methods and their replacements

Class	Deprecated Methods	Replacement Methods
`java.awt.Component`	`enable, disable`	`setEnabled`
	`show, hide`	`setVisible`
	`location`	`getLocation`
	`move`	`setLocation`
	`size`	`getSize`
	`resize`	`setSize`
	`bounds`	`getBounds`
	`reshape`	`setBounds`
	`preferredSize`	`getPreferredSize`
	`inside`	`contains`
	`locate`	`getComponentAt`
`java.awt.Container`	`countComponents`	`getComponentCount`
	`insets`	`getInsets`

Table 41. Depreciated methods and their replacements

Class	Deprecated Methods	Replacement Methods
`java.awt.Menu`	`countItems`	`getItemCount`
`java.awt.MenuBar`	`countMenus`	`getMenuCount`

Note: The JIS API TabbingManager.isFocusable method has been marked deprecated and was replaced by the isTabTraversalFocusable method.

Where Can Code Extension Be Performed

You can extend the Java code of the following environments:

- A single Subapplication
- The main window
- All the Application's Subapplications

Extending the Code of a Subapplication

The JIS Interface Server's Java Client creates a Java class for each Subapplication, as well as a class for the main window of the application. These classes form the `appls.<ApplName>.original` package, where `<ApplName>` is the application's name. You must not modify the classes of this package since they are recreated every time the Generate Runtime process is performed. The correct way to modify the Java code is to extend these classes; the inherited classes should be a part of the `appls.<ApplName>.user` package.

The following procedure should be used for each Subapplication that will be modified (the examples refer to a Subapplication called `LOGIN`):

To extend a Java code for a Subapplication called `LOGIN`:

1. In the `appls.<ApplName>.user` package create a class which will be an extension on the original class `appls.<ApplName>.original.LOGIN`. To do so, create a file called LOGIN.java in the `<InstallDir>\src\appls\<ApplName>\user` directory (by convention, the class name should have the same name as the Subapplication). This file will contain the following class definition:

```
package appls.<ApplName>.user;
import java.awt.*;
import cst.gwt.*;
public class LOGIN extends appls.<ApplName>.original.LOGIN {
}
```

2 Register this class in `appls.<ApplName>.user.JacadaStarter` by adding a line to its `registerUserGui` method:

```
addWindow(ApplPackage + "LOGIN");
```

This line will because the application to use the descendant class (`appls.<ApplName>.`**`user`**`.LOGIN`) instead of the original class (`appls.<ApplName>.`**`original`**`.LOGIN`). For more information about the `addWindow` method consult the section The JacadaStarter's addWindow Method.

3 Add the desired functionality to the class you created, as described in the following sections.

4 Compile your changes using the JACC batch file.

Note: Use this procedure whenever a specific Subapplication needs to be modified.

Extending the Code of the Main Window

Extending the code of the Main Window is useful when adding Java extensions to the tool bar. This is done exactly as described in the previous section—Extending the Code of a Subapplication—except for the following differences:

1 The Java file that you create in the `<InstallDir>\appls\<ApplName>\user` directory should extend the class: `appls.<ApplName>.original.MainWindow.`

2 Registering this class in `appls.<ApplName>.user.JacadaStarter` is done by adding the following line to its `registerUserGui` method:

```
addWindow(ApplPackage + "MainWindow");
```

Note: This should be the first call to `addWindow()`, before any other call to `addWindow()` to extend any of the Subapplications.

This line will cause the application to use the descendant class (`MainWindow` in the user directory) instead of the original class (`MainWindow` in the original one).

Extending the Code of All Subapplications

If you want to change the behavior of all Subapplications, you can modify the class `appls.<ApplName>.user.ApplSubApplWindow` since all Subapplication classes are its descendants. To do so

1. Add the desired functionality to the `ApplSubApplWindow` class.
2. Compile your changes using the JACC batch file.

   However, some modifications should be performed only once, when the application is started. These modifications are typically application-wide properties, and should be performed in the `appls.<ApplName>.user.JacadaStarter` class.

Note: The changes you make to the application's ApplSubApplWindow class influence only the main Application's Subapplications and not the Subapplications contained within its libraries.

Understanding the Generated Java Code

This section describes the functions that the original Subapplication classes perform. Since the users inherit these classes when extending a Java Client Subapplication, it is important to understand their functions and the ways to interface with them.

The subsequent sections provide examples that demonstrate the implementation of these principles.

Creating Controls

The Subapplication classes are responsible for the creation of the controls they contain, and the setting of each control's properties (fonts, colors, text, etc.). This function is performed inside the `createGUIControls` method, as can be seen in the Subapplication classes that JIS Interface Server creates. It is important to note that the Subapplication classes are containers, so their controls are added directly to them. No layout is used for the positioning and sizing of the controls—the coordinates and dimensions are the exact ones that were set in JIS Interface Server.

Any control that you wish to add to the Subapplication should be created and added in the `createGUIControls` method. Remember to call the super's method before doing anything else, otherwise the Subapplication will not work.

Example 36. Adding controls in the createGUIControls method

```
public void createGUIControls () {
   // First call the super's method
   super.createGUIControls();
   // Now add our own controls
   Label hello = new Label("Hello!");
   add(hello);
   hello.setBounds(50, 75, 100, 25);
}
```

Creating Logic Peers

In the JIS Interface Server, the GUI is separated as much as possible from the application's logic. This means that the controls are pure GUI, and contain no application logic. This logic is kept instead inside Logic Peer objects—each GUI control has a logic counterpart that connects it to the application as a whole, and to the JIS Server. These Logic Peers form the `cst.client.logic` package, and are created by the method `createLogicControls`. This method is called right after `createGUIControls`.

Normally, no changes should be made to this part of the Subapplication class.

Event Handling

Another important function of the Subapplication classes is the handling of events that happen inside them. The Java Client uses the Delegation Event Model which was introduced in JDK 1.1. In this model, controls are sources of different events, such as focus events, key events, mouse events, etc. Any object that wants to receive these events from a specific source must register itself with the source as a listener for a specific type of event. This also means that the listener object must implement one or more interfaces which define the methods that are called when an event occurs. For example, `java.awt.Button` sends an `ActionEvent` when it is clicked. An object that wants to receive these events must implement the `ActionListener` interface, and register itself with the button by calling the button's `addActionListener` method. When the button is clicked, the listener's `actionPerformed` method will be called. For details about the Delegation Event Model, please refer to the API of the `java.awt` and `java.awt.Event` packages.

In the Java Client, the Subapplication classes listen to events from the controls that they contain, and perform various functions according to the events received. The Subapplication registers itself with each control as a listener for different types of events in the `setControl` method.

The types of events vary according to the control's type:

- Focus events - all components.
- Mouse events - all components.
- Key events - all components.
- Action events - buttons, prompt controls and menu items.
- Item events - check box menu items.
- Table events- tables. Events generated by the Java Client's Table component.

The different event listener methods that the Subapplications implement perform the relevant functions. For example, clicking a button will result in a call to `actionPerformed`, and this method will result in sending the appropriate command to the JIS Server.

To handle events from a control in a Subapplication, you must override the appropriate listener method in the class of the Subapplication. For the default behavior to be performed, call the super's method from your method. The following examples will clarify this:

```
public void actionPerformed (ActionEvent evt) {
   if (evt.getSource() == myButton) {
      // Do something when myButton is clicked
      ...
   }
   else {
      // Handle the event in the default manner
      super.actionPerformed(evt);
   }
}
public void keyPressed (KeyEvent evt) {
   if (evt.getKeyCode() == KeyEvent.VK_HOME && evt.is-
   AltDown()) {
   // Do something when Alt+Home is pressed,
   // anywhere in the subapplication
   ...
   }
   // Always perform the default handling
   // (even when it's Alt+Home).
   super.keyPressed(evt);
}
```

There are only two cases in which you need to register the Subapplication as a listener with a specific control for a specific event type:

1. When the event type is not one of those listed above (for example, `ComponentEvent`).
2. When you added the control (not JIS Interface Server), and therefore the Subapplication does not recognize it.

Focus and Tabbing Management

Focus events enable the Subapplication to keep track of the input focus location. This task is delegated to a `TabbingManager` object, that knows which component within the Subapplication has the input focus. In addition, the `TabbingManager` keeps a list of the Subapplication's components, sorted by their tabbing order. This list is used when the user presses the tab key, in order to pass the focus to the next or previous control on the list. The focus is passed only to components that are visible, enabled, and defined as "focus traversable" (see `java.awt.Component.isFocusTraversable`).

The tab order of the components is set in the `setControl` method. In case you add components which you want to be a part of the tabbing order, you must add them to the TabbingManager's list. You can either replace one component by another or insert an additional component to the list. For example, the line

```
tabbingManager.addComponent(myButton, 10);
```

will add the component `myButton` as the tenth component on the tab-order list overwriting the component originally in that place. The line

```
tabbingManager.insertComponent(myButton, 10);
```

will insert the component `myButton` in the tenth place of the tab-order list. In addition, you can remove a component from the list. For example, the line

```
tabbingManager.removeComponent(myButton);
```

will remove the component `myButton` from the tab-order list. Make sure that the component returns `true` from `isFocusTraversable`. Another possibility is to locate your component after the location of another component. For example, to add `myButton` after the component `button17`, use:

```
int pos = tabbingManager.getComponentPosition(button17);
tabbingManager.insertComponent(myButton, pos + 1);
```

You may also wish a certain component or several components to be skipped over during the tabbing process. In this case, and assuming the component name is `myButton`, you should add the following line:

```
myButton.setFocusTraversable(false);
```

Keyboard Management

As was mentioned earlier, the Subapplication classes listen to key events from their components. These key events are used for three purposes:

1 Handling accelerators that were defined in JIS Interface Server.

2. Tabbing between components, as described in the last section.
3. Activating the default button, which is the button that should be pressed when the user presses Enter.

Any key-press is first looked up in the list of the Subapplication's accelerators. If it is found there, the server is notified that a specific accelerator has been activated. This means that if you want to block an accelerator, you should extend the `keyPressed` method (see example under "Event Handling" on page 201). Please note that the accelerator mechanism of the Java Client does not use the "menu shortcuts" defined in JDK 1.1, because they are too limited.

The default button is set by the `setDefaultButton` method. When the user presses ENTER inside a component that does not handle the ENTER on its own, the default button is activated. More specifically, it is pressed in `keyPressed` and released in `keyReleased`, and this results in its activation.

Examples of Code Extension

When writing a code extension, you must decide where to perform it. The following guidelines should cover most cases:

1. If the desired functionality should be present in all Subapplications, add the code extensions to the `appls.<ApplName>.user.ApplSubApplWindow` class.
2. If the code extension should be performed only once, when the application is first started, add it to the `appls.<ApplName>.user.JacadaStarter` or `appls.<ApplName>.user.Applet` classes.
3. If you wish to add some controls that will always be present in the application's main window (e.g. a toolbar), extend the `MainWindow` class.
4. Otherwise, extend only the specific Subapplications that you want to modify.

The appropriate method to create these extensions was discussed in the section Where Can Code Extension Be Performed. This section brings concrete examples of code extensions, that can be adapted for your specific needs.

Adding a Background Image

The `cst.gwt.GUIPanel` class, which is the ancestor of all Subapplications, supplies a method for displaying an image in its background. The image can be displayed in one of several modes, as described below. To set the background image for your Subapplication extend the Subapplication in the usual manner and inside its constructor or inside `createGUIControls` call the following method:

```
setBackgroundImage(spi, name, style);
```

The "`spi`" is the application's `SessionParmsInterface` object, which is usually the application's `JacadaStarter`. You get it by calling `getStarter()`.

The "`name`" parameter is the location of the desired image, in relation to the applet's codebase (for example, `IMAGE_DIR + "LOGO.gif"`, for an image that resides in the same directory as the rest of the application's images).

The "`style`" parameter should be one of the following:

- `PLAIN_BACKGROUND_IMAGE` — the image is displayed at the top-left of the Subapplication panel. This mode is suitable for images that are at least as large as the Subapplication.
- `CENTERED_BACKGROUND_IMAGE` — the image is displayed at the center of the Subapplication panel.
- `SCALED_BACKGROUND_IMAGE` — the image is scaled to cover the whole Subapplication panel. The image's proportions are not preserved, so this mode is suitable mainly for abstract background images.
- `TILED_BACKGROUND_IMAGE` — the image is tiled to cover the whole Subapplication panel. This mode is not recommended for small images, as their repetitive drawing would slow down the GUI considerably.

Example 37. Adding a background image

```
public void createGUIControls () {
   super.createGUIControl();
   setBackgroundImage(getStarter(),
                      IMAGE_DIR + "LOGO.gif",
                      CENTERED_BACKGROUND_IMAGE);
}
```

Note: When selecting a background image, consider its size in kilobytes—a large size might result in a long delay when the Subapplication is displayed for the first time.

Adding Action Buttons to a Subapplication

This section describes how to add action buttons to a specific Subapplication. Such buttons can perform various functions implemented in Java, such as the two following examples: a button that opens an HTML page in your Internet browser, and a button that plays an audio clip.

Perform the following to add action buttons to a Subapplication:

1. Add the desired buttons to the Subapplication using JIS Interface Server. You should give them descriptive names, such as `urlButton` and `audioButton`.
2. Using JIS Interface Server, perform the compilation procedure.

3. Perform the steps described in Extending the Code of a Subapplication, in order to inherit the Subapplication.
4. Let us assume that now you have a class in the user directory that inherits the Subapplication class you wish to modify. In this class, add the `actionPerformed` method as follows:

```
package appls.<ApplName>.user;

import java.net.*;
import java.awt.event.*;

public class LOGIN extends appls.<ApplName>.original.LOGIN {

public void actionPerformed (ActionEvent evt) {
   // Has the URL button been pressed?
   if (evt.getSource() == urlButton) {
      try {
      URL url = new URL("http://www.jacada.com");
      getApplet().getAppletContext().showDocument(url,
      "A");
      }
      catch (MalformedURLException e) {
      System.out.println("Malformed URL");
      }
   }
   // Has the audio clip button been pressed?
   else if (evt.getSource() == audioButton) {
      // The code assumes that the clip (in format .AU) is
   stored
      // in <InstallDir>/classes/appls/applname/au-
      dio/effect.au
      getApplet().getAudioClip(getApplet().getCodeBase(),
                     "appls/APPLNAME/audio/effect.au").play();
   }
   else {
      // Handle the event in the default manner
      super.actionPerformed(evt);
   }
  }
}
```

Compile your Subapplication using the JACC batch file.

Test the Subapplication to verify that the action buttons operate as specified.

The same method can be extended to include, for example, a button that launches a calculator window. Add the following lines to the above method (supposing the button you have added to the Subapplication is called `calcButton`):

```
   // Has the calculator button been pressed?
   else if (evt.getSource() == calcButton) {
      cst.misc.PocketCalc calc = new cst.misc.PocketCalc();
      calc.setLocation(100, 200);
```

```
        calc.show();
    }
```

Note: You can make such changes directly in Java, without adding components through JIS Interface Server. Such changes, however, that will not show in JIS Interface Server.

Adding Action Buttons to the Main Window Tool Bar

You can add action buttons to the main window's tool bar, by performing the same steps as described in the previous section, but with the following differences:

1. In JIS Interface Server, add the desired buttons to the application's GS_BAR, and not to a specific Subapplication.
2. Inherit the `appls.<ApplName>.original.MainWindow` class, and add the `actionPerformed` method there.

Querying a Button to Determine Its Characteristics

The following APIs have been added to the `GUIEmptyButton` class. The `GUIEmptyButton` class is the parent of all button types. These APIs let you extract information about a button control, such as its color, the thickness of its border, and more.

Obtain the Index of the Accelerator Character

The following code shows how to obtain the index of the accelerator character that is related to the button.

```
/* Return index of the accelerator char  */
public int getAcceleratorIndex()
```

Get the Button Color

```
/*  Returns the colors for drawing the button  */
protected Color getShade(int shade)
```

Get the Thickness of the "LineBorder" Button Border

```
/* Returns the thickness of LineBorder border */
protected int getLineBorderThickness()
```

Get the Thickness of the “Border” Button Border

```
/* Returns the thickness of the “Border” button border */
protected int getButtonBorderThickness()  /*
```

Get the Thickness of the “Dotted” button border

```
/*  Returns the thickness of Dotted  border  */
protected int getDottedBorderThickness()
```

Adding Bubble Help to Components

Every Subapplication class has a `GUIBubble` component, which displays the bubble help of the Subapplication’s components. Any component that implements the `cst.gwt.general.Describeable` interface can have a bubble help. Most of the Java Client components already implement this interface. In case you want to have a bubble help attached to a component that you've added, you must make sure that it implements the `Describable` interface. This interface defines only two methods:

```
public void setDescription (String description);
public String getDescription ();
```

In addition, the component has to be registered with the Subapplication’s `GUIBubble`, in the following manner:

```
bubble.addComponent((Describeable) comp);
```

> **Note:** This registration is not needed if the component was already given a bubble help text in JIS Interface Server.

The component's `getDescription` method should return the text to be displayed as its bubble help. This text does not have to be constant—it can change dynamically. For example, you can create an icon of a clock, whose bubble help displays the current date and time, or you can create a component that displays values, and whose bubble help gives a detailed explanation of the current value's meaning (a sort of context-sensitive help).
The bubble help text can be broken into several lines using the newline character ('\n'), as shown in the following line:

```
comp.setDescription("Type your age\n(in years) here.");
```

Example 38. Adding a Paste button

The following example shows how to add a Paste button into the main window’s toolbar. The button’s bubble help describes the exact text that will be pasted when the button is pressed:

```
package appls.<ApplName>.user;
import java.awt.*;
import java.awt.event.*;
import cst.gwt.*;
import cst.gwt.general.*;
public class MainWindow extends appls.<ApplName>.original.MainWindow


{
   PasteButton paste;

   public void createGUIControls () {
      super.createGUIControls();
      // create the paste button and add it to the toolbar
      paste = new PasteButton();
      toolbar.add(paste);
      paste.setBounds(300, 2, 50, 24);
      // listen for action events from the paste button
      paste.addActionListener(this);
      // register the paste button with the bubble help
      bubble.addComponent((Describeable) paste);
   }

   public void actionPerformed (ActionEvent evt) {
      if (evt.getSource() == paste) {
         // paste from the clipboard into the focused component
         cmdPaste();
      } else {
         super.actionPerformed(evt);
      }
   }

   // A button for pasting text. Its bubble help displays
   // the text that is currently in the clipboard.
   private class PasteButton extends GUIButton {
      public PasteButton () {
         super("Paste");
         // ensure that the button doesn't steal the focus,
         // otherwise the paste will fail.
         setFocusable(false);
      }

      // override the default getDescription() method
      public String getDescription () {
         // get the clipboard's contents
         String contents =
            ClipboardManager.getContentsAsString(this);
         // return the description accordingly
         if (contents == null) {
            return "No text to paste";
         }
```

```
            return "Paste \"" + contents + "\"";
        }
    }
}
```

Adding Animated Buttons to Subapplications

Using the `GUIAnimatedButton` class, you can create a cartoon-like button. Whenever the mouse pointer goes over the button, a sequence of images is displayed.

The `GUIAnimatedButton` class is part of the cst.gwt package.

The constructor of this class gets 4 parameters:

1. The `SessionParmsInterface` (usually `getStarter()`).
2. An array of image names. The first name is the name of the default image, the image that is displayed when there is no animation. The other names are those of the animation images.
3. The animation delay, in milliseconds.
4. Whether to draw a border for the button (use "False" if the images already have a border).

Another useful method is `setRepeat()`.By default, the animation sequence is displayed only once, whenever the mouse enters the button's area. By calling `setRepeat(true)`, you can tell the button to display the animation sequence repeatedly.

Example 39. Displaying an animation sequence repeatedly

```
public void createGUIControls () {
   super.createGUIControls();
   SessionParmsInterface spi = getStarter();
   String images[] = {"cstimgs/JISA.gif",
                      "cstimgs/JISB.gif",
                      "cstimgs/JISC.gif",
                      "cstimgs/JISA.gif"};
   GUIAnimatedButton button =
      new GUIAnimatedButton(spi, images, 250, true);
   button.setRepeat(true);
   add(button);
   button.setBounds(20, 20, 50, 50);
}
```

Updating Menu Items in Runtime

The Java client supports runtime activity of menu options such as checking/unchecking, enabling/disabling, and updating menu items text according to the host. These effects are usually achieved through the use of user-methods in JIS Interface Server.

However, for the check/uncheck menu items feature to work, you must extend the Java code and change these menu items' type, from regular menu items (`GUIMenuItem`) to a type that can be checked/unchecked (`GUICheckboxMenuItem`).

To do so:

1. Extend the desired Subapplication as described in the section Extending the code of a Subapplication.
2. Add the method `overrideGUIControls` and change the desired menu item's type.
3. Compile your Subapplication using JACC.bat, and register it in the JacadaStarter class by calling `addWindow`. See section "Extending the Code of a Subapplication" on page 198.

Example 40. Updating menu items during runtime

For a menu item called `View_HostScreen_18202`, extend the code as follows:

```
public void overrideGUIControls () {
   String label =
      ((GUIMenuItem) View_HostScreen_18202).getOrigi-
   nalLabel();
   View_HostScreen_18202 = new GUICheckboxMenuItem(label);
}
```

Defining Number and Length of Lines in Multi-line Edits

You can set by code extension the number of lines in the control, and the maximum length of each line. This is done by calling the method `setMaxLengths()`.

For example, if you want the control to have 3 lines with 20 characters each, call: `comp.setMaxLengths(new int[] {20, 20, 20});`

Doing so changes the control's behavior. It will now act very similarly to the way a multi-line field acts on the iSeries or Mainframe. For example, when you type and reach the end of one line, you will be automatically moved to the next one.

In addition, the text that the control sends to the Server will not contain any newline characters. Instead, each line will be padded with blanks up to the maximum length required by the user.

Selecting One Cell in Table Rows Using Right Click

By default, clicking with the right mouse button (RMB) on any cell in a table causes the entire row to be selected. However, some cases may require that only a specific cell within a row be selected when clicking the RMB.

To achieve this:

In the Java Client, extend the code of the Subapplication containing the table whose RMB behavior you wish to modify. To do so call the method `setSelectRowOnRMB` in the following manner:

```
public void createGUIControls () {
    super.createGUIControls();
    myTable.setSelectRowOnRMB(false);
  }
```

Note: `myTable` is the name given in this example. For each table you must provide the table name as given in JIS Interface Server.

Adding Content to a Table Cell

The `setCellValue` method enables you to insert information in editable Table cells. In the method's parameters you define the cell's location and the text you wish to insert, as follows:

```
public void setCellValue (int row, int col, String text)
```

Parameters

The following are parameters for the `setCellValue` method.

row Sets the value of a specific cell. The cell's row index is 1 based.

col Sets the value of a specific column. The cell's column index is 1 based. col is the original column number as defined in the original source code regardless of any runtime column reordering.

text The new cell value. Note that null value is not accepted.

Example 41. Adding contents to a table cell

Pressing a button will cause a given cell's numeric value to increment by 1. Note the use of the `getCellValue` method for retrieving a cell's content.

```
public void actionPerformed(ActionEvent evt) {
  if (evt.getSource() == myButton) {
     int val = new Integer(myTable.getCellValue(10,2)).
                                        intValue() + 1;
     myTable.setCellValue(10,2,""+val);
  } else {
    super.actionPerformed(evt);
}
```

Handling Table Selection Events: Enabling the List Menu

The `tableSelectionChanged` method is called (as a result of a TableEvent) each time the user changes the current selection in a table inside the Subapplication. It allows, for example, to disable the List menu when the current selection is not appropriate. This method was added to the Subapplication classes.

Note that this method is only called after selection changes that the user made, not after changes that originate from the server.

Below is an example that enables the List menu only when the table's selection includes whole lines. In this example, the application is called MYAPPL, the Subapplication is called MYSA, and the table is called MYTABLE:

```
package appls.MYAPPL.user;
import java.awt.*;
import cst.gwt.table.*;
public class MYSA extends appls.MYAPPL.original.MYSA {
   public void tableSelectionChanged (TableEvent evt) {
      updateListMenu();
   }
   public void windowReadyForAction () {
      updateListMenu();
   }
   void updateListMenu () {
        // Get the first selected cell in the table.
      TableSelection ts = MYTABLE.getSelection();
      Point cell = ts.getFirstSelectedCell();
        // Enable the List menu only if a whole row is selected.
```

```
        // In such a case, the cell's column is 0.
      boolean enabled = (cell != null && cell.x == 0);
      if (List.isEnabled() != enabled) {
         List.setEnabled(enabled);
      }
   }
}
```

For more details, consult the javadoc of the following classes:

`cst.gwt.table.TableEvent`

`cst.gwt.table.TableSelection`

Manipulating Host Originated Data

You can display in the Subapplication information based on runtime field data. This additional enhancement to the Subapplication is achieved by extending the Java code and overriding the ***windowDataReady*** method.

The `windowDataReady` method is automatically invoked whenever a new Subapplication is called or the current Subapplication is refreshed. The method is activated after the window's fields have been filled with the updated information from the host and just before the Subapplication is displayed. As a result, actions that are performed in this method will have taken effect when the Subapplication is displayed, reflecting the updated runtime information.

Another method that is automatically called is `windowReadyForAction`. It is always invoked after `windowDataReady`, when the Subapplication is displayed. This method can be used , for example, to set the focus on a specific control in the window. However, any data manipulation should probably be performed in `windowDataReady`, to prevent the user from actually seeing the changes that are being made.

You can override the `windowDataReady` method in the class of a given Subapplication to enable such enhancements as:

- Associating a bitmap with the value of an output field. (You will have to set the name of the bitmap file to be displayed in the `windowDataReady` method).
- Playing an audio-clip whose name is given in one of the fields.
- Changing text colors according to the meaning of the text (e.g., a label that displays the name of a color can be colored according to its contents).
- Automatically setting the value of a field (possibly based on the current values of other fields). This possibility is concretized with the next example:

Note: You must not perform any blocking or time-consuming operations in the `windowDataReady` method, since the client does not process any further requests from the server, until this method returns. You may relax this limitation by performing the time-consuming operation in a separate thread which will be started in the `windowDataReady` method.

Example 42. Manipulating host originated data

In this example, the Total label is added to a Subapplication featuring a table. The Total label shows the sum of the values appearing in the cells in the second column from the left. The sum is displayed in the Total label as soon as the new Subapplication is shown to the user.

The following class will create the Total label:

```
package appls.<ApplName>.user;
import cst.gwt.*;

public class MySubAppl extends appls.<ApplName>.original.SubAppl {
   public GUIMultiLineLabel total;

   // Create the window's GUI controls
   public void createGUIControls () {
      // Call createGUIControls that was generated during compilation
      super.createGUIControls();
      // Create the Total label
      total = new GUIMultiLineLabel("Total:");
      // Add the Total label to the window
      this.add(total);
      // Set size and location of the Total label
      total.setBounds(30, 270, 100, 20);
   }

   // The windowDataReady method is called after the
   // subapplication's controls get their data and before the
   // subapplication is displayed
   public void windowDataReady () {
      // the sum of the column's cells
      int sum = 0;
      // the number of rows in myTable
      int rowsNo = myTable.getNumRows();
      // loop over table's rows
      for (int i = 1; i <= rowsNo; i++){
        // get the contents of cell(i, 2) and add it to the sum
        // note that the row and column indices start with 1
         sum += Integer.parseInt(
                   (myTable.getCellValue(i, 2).trim()));
      }
```

```
        // display the sum in the Total label
        total.setLabel("Total: " + sum);
    }
}
```

As a result of this code enhancement, the Total label is added at the bottom of the table's second column, as illustrated below.

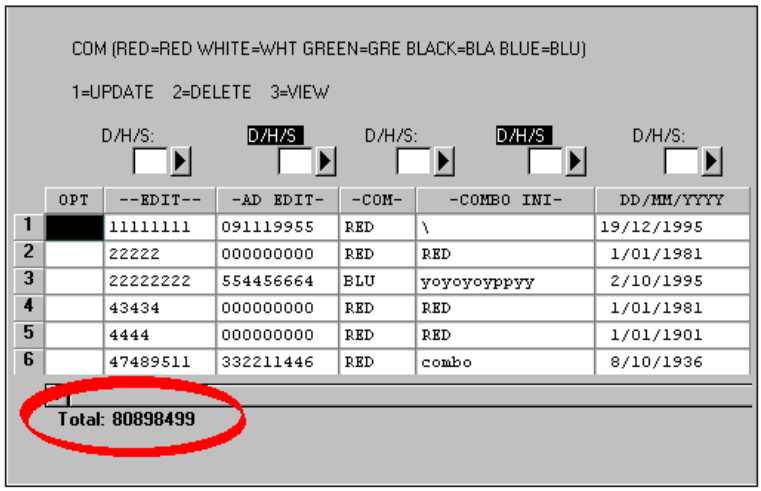


Figure 33. Total label added to the bottom of the table's second column

The Java Client RMB Floating Menus Support

Controls created in JIS Interface Server are supplied with a default response to right mouse button clicks. The Java client supports the default RMB behavior as defined in JIS Interface Server.

Clicking the RMB displays a popup menu containing:

- The List menu for tables
- The Edit menu for editable controls (edit, prompt, date, combo...)
- The Commands menu for all other controls.

In addition, you can define your own customized floating menu and attach it to a right mouse button click. Attaching a user-defined floating menu to a control requires a Java code extension.

To create a user defined floating menu:

1 In JIS Interface Server, create the floating menu and assign functionality to each of the menu's items. See Floating Menus in *JIS Interface Server: Basic User's Guide*.

2 In JIS Interface Server, compile the application. Floating menus that were defined in the application are now converted to Java source inside the parent-class of all Subapplications,

`appls.<ApplName>.original.ApplSubApplWindow`. Each user-defined floating menu has its own method that causes it to be displayed. This method is called `createMenu_MENUNAME` where MENUNAME is the floating menu's name that was given in JIS Interface Server.

3 Extend the Java code to display floating menus using the right mouse button click, as described in the following section.

Modifying the Default Floating Menus Behavior

By using a code extension you can change the floating menu behavior that exists in the Java Client. You may want to add floating menus that were defined in JIS Interface Server, or you can even apply your own logic as to which floating menu to display.

Displaying a Floating Menu for a Specific Control Type

To specify a floating menu that will be displayed over a specific control type, override one or more of the following methods:

```
UserRMBTable()
UserRMBPrompt()
UserRMBSpin()
UserRMBcombobox()
UserRMBDate()
UserRMBEditBox()
UserRMBStatic()
UserRMBRadioButton()
UserRMBCheckBox()
UserRMBGroupBox()
UserRMBButton()
UserRMBFrame()
UserRMBWindow()
```

In the overridden method you should add a call to the generated `createMenu_...()` method.

Example 43. Displaying a floating menu for a specific control type

```
Displaying a menu called MYMENU over all combo boxes can be done by:
```

```
protected void UserRMBcombo box(){
   createMenu_MYMENU();
}
```

Alternatively, you may call the `copyMenuFromMenubar()` method to display a floating menu which is copied from the menu-bar.

Example 44. Displaying a floating menu copied from the menu-bar

Displaying a floating menu which is copied from the menu "MyMenu" in the menu-bar, over buttons:

```
protected void UserRMBButton(){
   copyMenuFromMenubar("&MyMenu");
}
```

Displaying a Floating Menu that is Not Attached to Any Control

You can also apply your own logic (not necessarily based on the component type) in order to decide which floating menu to display. To do so you need to override the `activateUserRMBMethod()` method.

Example 45. Display a floating menu not attached to a control

Displaying a menu called MYMENU2 over all the enabled components can be done by:

```
protected void activateUserRMBMethod () {
   if (lastRMBTarget.isEnabled()) {
      createMenu_MYMENU2();
   }
   else
   {
      super.activateUserRMBMethod();
   }
}
```

Note: In all RMB methods, the extended Java code can identify the component over which the RMB was clicked by looking at the value of the protected variable: `lastRMBTarget`

Changing the Titles of Tab Control Folders During Runtime

Using code extension, you can update tab titles during runtime. This is enabled through the following method included in the `GUITab` class:

```
public void setFolderName (int index, String name)
```

Note: The index is zero-based.

For example, suppose the variable name of a tab control is Var17, and you wish to change the title of its second folder according to the text in a control represented by the variable Var18. To do so, extend the Subapplication class, and override the `windowDataReady` method:

```
public void windowDataReady () {
   Var17.setFolderName(1, Var18.getText());
  }
```

Displaying Message Boxes

The standard message boxes in the Java Client can be used by your code extensions to display messages to the users or receive input from them. To do that, use the `messageBox` method in the `JacadaStarter` class. This method is defined as follows:

```
int messageBox (String text, String  title,
                int type, boolean modal)
```

The method's parameters are:

text The message to display.

title The message window's caption.

type The type of message box to use. The possible types are defined in `cst.client.ApplWin.CSTInfoDialog`, and include the following:

```
OK_DIALOG
    YES_NO_DIALOG
    OK_CANCEL_DIALOG
    YES_NO_CANCEL_DIALOG
```

modal Whether the message box should block input to all other windows until it is dismissed. If this parameter is omitted, it is assumed to be false (meaning not to use a modal message box).

Note: In case the method is invoked from within the AWT events thread (e.g. as a response to a keyboard or a mouse event), a modal dialog must be used. Otherwise, the events thread will be blocked and the application will hang.

After the user clicks on one of the buttons in the message box, an integer code is returned according to the button which was clicked. The possible return codes are:

```
cst.client.ApplWin.DialogAnswer.OK
cst.client.ApplWin.DialogAnswer.CANCEL
cst.client.ApplWin.DialogAnswer.YES
cst.client.ApplWin.DialogAnswer.NO
```

Assume that some Subapplication displays a database record and allows to user to delete it. To prevent accidental deletion, you can display a message box asking for confirmation before the record is actually deleted. Here's how to achieve this:

```
public void actionPerformed (ActionEvent evt) {

      // Check if this is the "delete record" button
      if (evt.getSource() == deleteRecord) {
         // Ask for confirmation
         int answer = getStarter().messageBox(
            "Are you sure you want to delete this record?",
            "Delete Record Confirmation",
            CSTInfoDialog.YES_NO_DIALOG, true);
         // Do not delete the record if the user answered "no"
         if (answer == DialogAnswer.NO) {
            return;
         }
      }
      super.actionPerformed(evt);
}
```

Creating Custom Validity Checks

The Java Client allows you to devise custom validity checks which will test data entered by the user for validity (in addition to the built-in validity checks that you can set in JIS Interface Server). This can be done by modifying the automatically-generated `appls.<ApplName>.user.ValidityCheck` class. As an example, we will create a custom validity check for testing whether the contents of a field is a valid time in an HH:MM format.

Creating Custom Validity Checks is a three step process:

1. Adding the Validity Method
2. Writing the Validity Check
3. Using the Validity Check

Adding the Validity Method

Decide on a name for the custom validity method, and check for it in the `checkValidity` method of the `appls.<ApplName>.user.ValidityCheck` class.

Example 46. Adding the validity method

In this example, the name of the validity method will be “ValidTime”.

```
public void checkValidity (String validityMethod,
                           int fieldLengthInBytes,
                           String fieldText,
                           int textLengthInBytes) {
   if (validityMethod.equals("ValidTime")) {
      checkValidTime(fieldText);
   }
   else {
      super.checkValidity(validityMethod, fieldLengthInBytes,
                          fieldText, textLengthInBytes);
   }
}
```

Writing the Validity Check

Now we need to implement the method that actually tests the user-entered data for validity. If the method finds the data valid, it should call `setValid` to mark the text as valid. Otherwise, it should call `setInvalid`. The `setValid` and

`setInvalid` methods receive the number of characters that were found to be valid until the first error. In addition, `setInvalid` receives an error message which will be shown to the user.

Example 47. Writing the validity check

Here the validity check is fairly simple—it finds the position of the colon character, and checks that there are numbers to its left and right, and that these numbers are in the correct range.

```
void checkValidTime (String text) {
String message = "value must be in HH:MM format";

// Find the colon character
int pos = text.indexOf(':');
   if (pos < 1) {
      setInvalid(0, message);
      return;
   }
// Get the hours and minutes
   int hours = -1;
   int minutes = -1;
   try {
      hours = Integer.parseInt(text.substring(0, pos));
      minutes = Integer.parseInt(text.substring(pos + 1));
   }
   catch (NumberFormatException e) {
   }
   // Check that they're in the correct range
   if (hours < 0 || hours > 23 || minutes < 0 || minutes > 59) {
      setInvalid(0, message);
      return;
   }
// If we got here, the text is valid
   setValid(text.length());
}
```

Using the Validity Check

Now that we have a new validity check, we can apply it to any field in any Subapplication. The way to do this is through the `setValidityMethod` of each control's logic peer. This method should be called inside the `createLogicControls` method, after the logic peers have been created and initialized.

Example 48. Using the validity check

Suppose that we have a text-field called `timeField` in some Subapplication. We can now extend that Subapplication in the usual manner, and apply the "ValidTime" validity method to this field:

```
public void createLogicControls () {
   super.createLogicControls();
   timeField_peer.setValidityMethod("ValidTime");
}
```

Application-Wide GUI Settings

As was noted previously, some aspects of the application's GUI are relevant to the whole application, and not to specific Subapplications. If you wish to modify any of these aspects, do so close to the start of the application, before any GUI has been created and displayed. This means that such settings should be made in the user's Applet (in its `init` method) or in the constructor of the `JacadaStarter` class.

> **Note:** Since the settings are kept in `static` variables, they will apply to all Java Client applications running on the same JVM.

The Application's Color Scheme

The class `cst.gwt.general.Palette` defines several colors which are based on `java.awt.SystemColor`. These colors are used by the Java Client components to draw themselves. If you change any of these palette colors, the components will use the colors that you chose. You can change specific colors such as `Palette.control` or `Palette.windowText`, or you can call the `assignColors` method to make the Java Client use shades of your preferred color.

Example 49. Application's color scheme

```
// Use shades of gray as the color scheme
Palette.assignColors(Color.lightGray);
// Use blue on yellow for editable controls, such
// as text fields, combo boxes and check boxes
Palette.window = Color.yellow;
Palette.windowText = Color.blue;
```

These settings should be made early in the execution of the applet - the `init` method of the user's applet is a good place.

Please note that the palette colors are not updated dynamically. That is, changing the system colors of the client computer will not affect any Java Client applications that are already running.

For more information, please refer to the API of `cst.gwt.general.Palette` and `java.awt.SystemColor`.

Multi-Character Search in Combo Boxes

Usually, typing a character in a combo box highlights the next entry that begins with that character. A more advanced search mechanism allows the user to select the exact entry in the combo list by typing the second letter, third letter etc. of the entry.

For example: a combo box contains the entries 'Cell', 'Cellular', 'Collision', 'Cosmic', etc. When you use the regular search mechanism, typing C, E, L, L will highlight the first entry beginning with C, then the first entry beginning with E, and so on. When you use the multi-character search mechanism, typing C, E, L, L will highlight the first entry beginning with C, E, L, L - 'Cell', or 'Cellular'.

The Java Client drop-down list combo boxes support the multi-character search mechanism. In order to it, add the following line inside the `init` method of the user's applet:

```
cst.gwt.GUIcombo box.setSelectionByPrefixDelay(###);
```

'###' is the number of milliseconds that the user will need to wait before he can start typing a new entry. For example, if the user typed CE and got the entry 'Cell', he will have to wait a few seconds before he can type COS and get 'Cosmic'. 1500-3000 milliseconds are a reasonable delay. A way of bypassing the delay is to press Delete or Backspace, after which the user can immediately type a new entry. Also, if the user selects an item using the mouse or using Up/Down/Page_UP/Page_Down/Home/End, he can start typing a new entry without waiting.

Data Sharing between Client and Server

The Java Client development environment includes a varpool that enables the Java client and Server to share information. The Server, using methods defined in JIS Interface Server, and the Client, using methods included in code extensions, can write variables to, or retrieve and delete variables from this common varpool.

Manipulating the Varpool from the Server Using Methods

To manipulate information in the varpool, the Server uses methods defined in JIS Interface Server. JIS Interface Server includes the following three DoMethods:

`WriteSharedUser Variable`	To write information to the varpool. This DoMethod requires two parameters: Value: The variable's value Key: The variable's name Returns: 1 for success, 0 for failure
`GetSharedUser Variable`	To retrieve information from the varpool. This DoMethod requires the following parameter: Key: The variable's name Returns: Null or the variable's name
`DeleteSharedUser Variable`	To delete information from the varpool. This DoMethod requires the following parameter: Key: The variable's name Returns: 1 for success, 0 for failure

Use these DoMethods to define methods according to the activity you wish to perform in the varpool.

Example 50. Manipulating the varpool from the server using JIS Interface Server methods

Example of how to use these methods in JIS Interface Server

The following code writes the variable "Name" and the value "Beatrice" to the varpool:

```
DoMethod: Receiver: `System` Method: WriteSharedUserVariable Parms: [`Name`,
`Beatrice`]
```

Manipulating the Varpool from the Client Using Code Extensions

The Client manipulates information in the varpool through code extensions. The following three methods are included within the `JacadaStarter` class:

`void writeSharedUserVariable (String key, String value)`	To write information to the varpool. This method requires two parameters: Value: The variable's value Key: The variable's name
`String getSharedUserVariable (String key)`	To retrieve information from the varpool. This method requires the following parameter: Key: The variable's name
`void deleteSharedUserVariable (String key)`	To delete information from the varpool. This method requires the following parameter: Key: The variable's name

Example 51. Manipulating the varpool from the client using code extensions

Example of how to use these methods within the code extension

The following code writes the variable "Name" to the varpool, retrieves the variable "Name" from the varpool, and deletes the variable "Name" from the varpool:

```
...
public void createGUIControls () {
      super.createGUIControls();
      JacadaStarter starter = getStarter();
}
public void actionPerformed(ActionEvent evt) {
      if (evt.getSource() == Var45) {
           starter.writeSharedUserVariable("Name","Beatrice");
      }
```

```
        else if (evt.getSource() == Var46) {
             String name = starter.getSharedUserVariable("Name");
        }
        else if (evt.getSource() == Var47) {
             starter.deleteSharedUserVariable("Name");
        }
        else {
             super.actionPerformed(evt);
        }
}
```

Note: Changes made to the user variables on the Server are sent to the Client the next time a Subapplication is shown or refreshed. Changes made to the user variables on the Client are sent to the Server upon the next action (button click, menu item click, etc.) This means that whenever a Subapplication is shown or refreshed, the Client and the Server hold the same shared user variables.

The JacadaStarter's addWindow Method

The `addWindow` method should be called when you wish to replace an existing Subapplication with a new enhanced version of the GUI. When `addWindow` is called, the class it refers to is loaded by the JVM.

The `addWindow` method exists in three forms:

1. For a standard replacement of a Subapplication with a user-GUI you should use:

   `addWindow(String className);`

   In this standard form, the `addWindow` method requires only the class name.

   Example52. Standard replacement of a Subapplication with a GUI

   ```
   addWindow(ApplPackage + "LOGIN");
   ```

2. If you also want to preload the class during the initialization, you should use:
   `addWindow(String className, boolean preload);`

Example53. Preloading a class during initialization

```
addWindow(ApplPackage + "LOGIN", true);
```

Note: The preloading process involves, besides loading classes, various initialization operations on the window to save time when displayed for the first time.

3 For special cases where a large number of user GUI classes should be registered, and it is undesirable to load all of them into the JVM during initialization, use:

```
addWindow(String libraryName, String windowName,
                      String className, boolean preload);
```

This form of `addWindow` does not load the class into the JVM to determine the names of the library and the window. Instead, it accepts the libraryName and windowName values as parameters from the caller. LibraryName should be an empty String ("") when the relevant Subapplication is not in any library.

Note: The windowName and libraryName should be written in upper case. For example: `addWindow("", "LOGIN", ApplPackage + "LOGIN", true)`

Launching the Java Client from an Applet

The following applet launcher classes and examples are retained as legacy API reference. Browser applet support was retired in JIS 9.2.4. For JIS 9.3, launch the Java client through Java Web Start / JNLP or as a Java application, and recompile customized client and server extensions after upgrading.

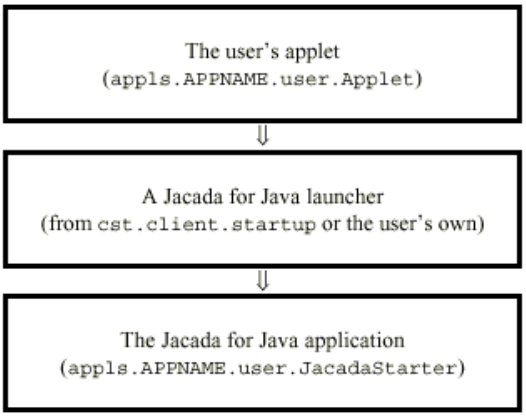


Figure 34. Launching the Java client from an applet

The applet creates the launcher by calling its `init` method. The launcher creates the `JacadaStarter` object by calling its `init` method.

During the compilation process, a skeleton for the applet class is created in the user's package directory. This applet uses the default launchers that are a part of the `cst.client.startup` package. The user can modify this class to fit his needs, and can replace the default launchers with his own launcher. The process of creating a new Java Client launcher is described in detail later this chapter.

The Java Client Launchers

The `cst.client.startup` package includes two different launchers which are used by the standard applet, as created by the compilation process:

- `JacadaBasicLauncher` – a standard launcher that immediately runs the application and displays messages from it.
- `JacadaLoginLauncher` – a more complicated launcher that requests the user to enter his username, password and/or profile name before starting the application. The choice of fields that will be displayed to the user is configurable through the applet's HTML parameters.

In addition, this package contains two other classes:

- `JacadaLauncherInterface` – an interface that must be implemented by all Java Client launchers. It defines a small set of services that the application expects to receive from the launcher.
- `JacadaLauncherCreator` – a class that creates a `JacadaLoginLauncher`, customized according to the appropriate HTML parameters.

Customizing a Launcher

It has been mentioned before that an existing launcher can be customized. The following Java class is an example of how to extend an existing launcher's code:

```
package appls.DEMO.user;

import java.awt.*;
import java.awt.event.*;
import cst.client.startup.*;
import cst.gwt.*;

public class MyLauncher extends JacadaLoginLauncher {

   GUIButton quitButton;

/**
   * Constructs a new MyLauncher.
   */
   public MyLauncher () {
      // Call the JacadaLoginLauncher constructor, and request
      // only the profile name from the user. The default
      // profile will be the "Guest" profile.
      super(false, false, true, "", "Guest");
   }

/**
      * Initializes the launcher.
      * @param applet the applet.
      * @param applicationClass application class full name.
      * @return false if initialization fails.
   */
   public boolean init (java.applet.Applet applet,
                        String applicationClass) {

   if (!super.init(applet, applicationClass)) {
         return false;
      }

      // Create the quit button
      quitButton = new GUIButton("   Quit   ");
      quitButton.setFont(button.getFont());
      quitButton.addActionListener(this);
      // Add it to the original button's container
      button.getParent().add(quitButton);
```

```
        // Change the colors of the message
        message.setForeground(Color.cyan);
        message.setBackground(new Color(0, 0, 70));

        return true;

    }

    /**
     * Handle action events - quit when quitButton is pressed.
     */
    public void actionPerformed (ActionEvent evt) {
        if (evt.getSource() == quitButton) {
          getParent().remove(this);
          return;
        }
        super.actionPerformed(evt);
    }
}
```

Example 54. Extending the JacadaLoginLauncher

In this example, the `JacadaLoginLauncher` was extended. Its appearance and functionality have been modified in the following manner:

- A default profile name was set.
- A Quit button was added.
- The message color was changed to cyan on a dark blue background.

The changes are presented in the picture below:

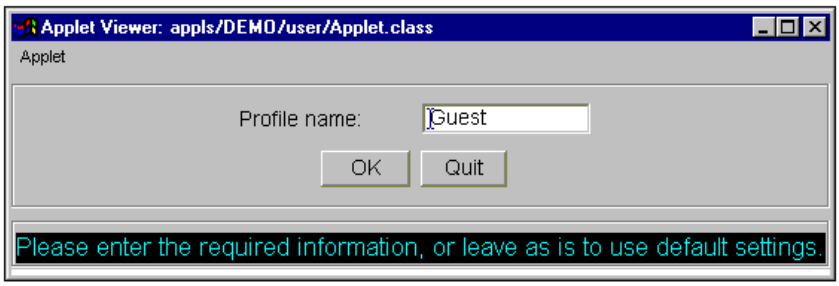


Figure 35. Extending the JacadaLoginLauncher

Note that `button` is a field of `JacadaLoginLauncher`, and that `applet`, `message` and `messageBox` are fields of `JacadaBasicLauncher` (which is the parent of `JacadaLoginLauncher`).

To make the applet use `MyLauncher` instead of the default launchers, make the following change in `appls.<ApplName>.user.Applet`.

Instead of

```
launcher = JacadaLauncherCreator.createLauncher(
              (java.applet.Applet) this);
```

write

```
launcher = new MyLauncher();
```

After compiling the `Applet` and `MyLauncher` classes (using JACC.BAT), the applet will use the new customized launcher.

Controlling the Java Client Application

The `JacadaStarter` class supplies a set of methods that retrieve information concerning the state of the Java Client application and send commands to it. The Java programmer can use this set of methods to “remote-control” the Java Client – to move between Subapplications and fill their input fields in an automated manner. This is done by incorporating these methods in powerful scripts that simulate users’ actions.

This section discusses the methods that the `JacadaStarter` provides for this purpose.

Methods for Controlling the Java Client Application

waitForFirstWindow

The first step in controlling the Java Client application is to make sure that it has started. Calling this method locks the current thread until the JIS Interface Server displays the first Subapplication. In case the method is called after the application has already started, it returns immediately.

Generally, methods that cause threads to wait until an action is completed should not be called from the main thread of the applet, as this would cause the applet's user interface to temporarily lock. Instead, a separate thread should be started and all the waiting can be done by it, allowing the applet to continue its normal execution.

Note: To use this method, you must have AutoStart enabled in your application. You can turn it on from the Option menu of the Windows runtime application.

getCurrentPanel

This method returns a `cst.gwt.GUICSTPanel` object. This is a subclass of `java.awt.Panel`, which is the last Subapplication displayed. Since all components in a Subapplication class are `public`, you can refer to specific members and modify their values by casting this object to the correct Subapplication class.

Example 55. SetCurrentPanel

For example, for a Subapplication called LOGIN that has members called `userField` and `passwordField`, you can write the following code:

```
GUICSTPanel currPanel = application.getCurrentPanel();
   if (currPanel instanceof LOGIN) {
      LOGIN login  = (LOGIN) currPanel;
      login.userField.setText("guest");
      login.passwordField.setText("welcome");
   }
```

activate

This method allows you to activate buttons, menu items and accelerators in the current Subapplication. This method comes in two variations.

In the first format, the method is given a component (a button, a prompt control, or a menu item), and it simulates the activation of that component. In addition, the method can be instructed to block the executing thread until the action is completed by passing `true` in the second argument of the call. This is needed for cases in which you wish to do something in the new Subapplication that will appear as a result of the action. You should be careful when instructing `activate` to block; do not block the execution of `activate` threads, such as the `awt` threads.

Example 56. activate

For example, let us assume that the LOGIN Subapplication from the previous example also has a button called `okButton`:

```
// "Press" the button, and wait for the next sub-application
   application.activate(login.okButton, true);
// Now the sub-application is supposed to change to MENUS...
currPanel = application.getCurrentPanel();
if (!(currPanel instanceof MENUS)) {
```

```
    // We're not where we expect to be! Maybe the Login
    // failed?
  }
```

Note: Only components which were created in JIS Interface Server can be activated by this method. In case there are components that have been manually added to the Subapplication, you must devise other ways in which to activate them (invoke their appropriate method).

The second format of the `activate` method simulates the action of an accelerator. It takes a key-code and its modifiers (as defined in `java.awt.event.KeyEvent`) and sends the key-press to the Subapplication. Here too, you can lock the current thread until the action is finished. Continuing with the example above, we will now send a key-press to the MENUS Subapplication:

```
import java.awt.event.KeyEvent;
. . .
// Press Ctrl+F1 and don't wait - the user will go on from here
application.activate(KeyEvent.VK_F1, KeyEvent.CTRL_MASK, false);
```

hideMainWindow

Calling this method with a `true` argument causes the application's main window, and any Subapplications displayed within it, to remain hidden. This allows you to move between Java Client Subapplications without the user seeing the process, or to temporarily hide the application from the user's view. Note, however, that any application popup windows or dialog boxes will be displayed.

Calling the method again with a `false` argument restores the main window to view.

This method may be called even before the first window of the application appears. This is useful when skipping the Login screen.

Note: You must not perform any locking instructions in the windowDataReady and windowReadyForAction methods with a true argument. This may cause the application to hang.

getMainWindow

This method returns a reference to the GUICSTPanel object that represents the main window of an application. This allows for activities to be performed on the main window, such as controlling menus and the status bar.

For example the following code will cause a message to be displayed in the status bar:

```
getStarter().getMainWindow().displayMessage("this text will appear on the
bottom of the screen");
```

waitForNextWindow

This method locks the current thread until a new window arrives from the server. The method returns immediately when it does not expect any new window.

isUserActionAllowed

Indicates whether a user action can be performed. The method returns FALSE to indicate that the client is waiting for the server to reply and that no user action will be performed while waiting. Otherwise the method returns TRUE.

Code Examples

This section contains code examples for your use.

Example 57. Skipping the login screen

This code example skips the login screen.

> **Note:** While debugging the code it is useful to see the GUI. You should add the `hideMainWindow(true)` only after the rest of the code is working as it should.

This example demonstrates the following points:

- The basic technique of using an ApplicationController—a separate thread that performs the waiting in order not to block the applet's main thread.
- Hiding the GUI window from the user, and redisplaying it.
- Entering values into TextFields in the GUI (The Login window).
- Hitting a button on the GUI window and waiting for the action to finish.

```
...
import appls.<ApplName>.original.*;
...
public class Applet extends java.applet.Applet {
   JacadaLauncherInterface launcher;

   public void init () {
```

```
      ...
      // Create and initialize the Java Client Launcher
      launcher = JacadaLauncherCreator.createLauncher(this);
      launcher.init(this, "appls.<ApplName>.user.JacadaStarter");
// Run the application controller in a different thread.
      // Otherwise, the applet will become stuck until the end
      // of the process.
      ApplicationController ac = newApplicationController(this, launcher);
      new Thread(ac).start();
   }
}
class ApplicationController implements Runnable {
   java.applet.Applet applet;
   appls.<ApplName>.user.JacadaStarter application;

   public ApplicationController (java.applet.Applet applet,
                                 JacadaLauncherInterface launcher) {
      this.applet = applet;
      application = (appls.<ApplName>.user.JacadaStarter)
                    launcher.getApplication();
   }

   public void run () {
      // Hide the GUI (we don't want the user to see the
      // login screen at all)
      application.hideMainWindow(true);

      // Wait for the first window to appear
      // (The application must have "auto-start" enabled)
      application.waitForFirstWindow();

      // Enter the user & password and hit "OK"
      LOGIN login = (LOGIN) application.getCurrentPanel();
      login.userField.setText("guest");
      login.passwordField.setText("foobar");
      application.activate(login.okButton, true);

      // Redisplay the GUI when reaching the screen which is
      // after the login
      application.hideMainWindow(false);
   }
```

```
}
```

Example 58. Manipulating information, navigation and acceleration

This code example demonstrates the following points:

- How to extract information from the GUI.
- How to perform smart navigation, based on the screen that appears on the host.
- How to invoke an accelerator (F3) on the GUI and wait for the action to finish.
- How to enter data into the GUI windows, using values that were given as parameters of the applet in the html page.

```
import java.awt.event.KeyEvent;

       ...

      LOGIN login = (LOGIN) application.getCurrentPanel();

      // Extract a field from the Login screen
      // Use getText() to get the value of a TextField field
      // ("edit" in JIS Interface Server)
      // Use getLabel() to get the value of a MultiLineLabel
      // field ("static" in JIS Interface Server)
      String systemName = login.systemNameField.getLabel();
      System.out.println("The name of the system is: " +
                         systemName);

      // Enter the user & password and hit "OK"
      login.userField.setText("guest");
      login.passwordField.setText("foobar");
      application.activate(login.okButton, true);

      // Skip the optional MESSAGE subappl
      Panel panel= application.getCurrentPanel();
      if (panel instanceof MESSAGE) {
         // Invoking the F3 accelerator defined in the GUI:
         application.activate(KeyEvent.VK_F3, 0, true);
      }

      // Fill in the data for the query using the ItemNum
      // HTML parameter of the applet:
```

```
     QUERYSER queryser  = (QUERYSER)
 application.getCurrentPanel();
     queryser.itemNumField.setText
                         (applet.getParameter("ItemNum"));
     application.activate(queryser .okButton, true);
```

Example 59. Waiting

This code example demonstrates how to wait for the user to hit a certain button in the current GUI window. The example refers to a Subapplication called RESULT.

```
// Wait for the user to hit OK
RESULT result = (RESULT) application.getCurrentPanel();
try {
   synchronized(result) {
      result.wait();
   }
} catch (InterruptedException e) { }

// The rest of the code (will be executed after the user
// hits OK)
...
```

In order to release this thread from its waiting status, you should extend the RESULT Subapplication class (as described in the section Extending the Java code), and add the following code to it:

```
public void actionPerformed (ActionEvent evt) {
         if (evt.getSource() == okButton) {
            synchronized(this) {
               this.notify();
            }
         }
         super.actionPerformed(evt);
      }
```

Implementing Localization Using the Java Localization API

The following methods are implemented in the JacadaStarter class.

Loading Specific Strings from Resource Files

`public String getLocalizedCstString(String key)`

Returns the general string resource (pre-defined by JIS) for the specified key in the current language. Useful when wishing to use a string that exists in the JIS resources, and not in the translated resource.

`public String getLocalizedString(String library, String key)`

Returns the string resource for the specified library and key (original language) string in the current language. If the resource file is not cached, it is located and loaded first.

Initialization of Language Localization

`public void setLocale(java.util.Locale loc)`

Sets the language of the UI. Should be called before the first window is created. Call the method within the `init()` method of the `appls.<ApplName>.user.JacadaStarter` class. The `init()` method is called just before returning from the call to JacadaLauncherInterface init (Applet, String) in the applet.

Example 60. Initialization of language localization

```
protected void init() {
   setLocale(Locale.FRENCH)
   start();
}
```

Changing the Language in a Running Application

The `JacadaStarter.SetLocale` method can also be used to change the language of the locale within a running application. In this case, the code line must be inserted in the location you wish the switch in languages to take place.

In order to allow the dynamic switch in languages, when the caching mechanism of previously used Subapplications has been applied on the application, the `clearCachedWindows` method has been introduced. This method clears the

cache from all saved Subapplications, thereby allowing the `SetLocale` method to take effect. The `clearCachedWindows` method must be inserted before `SetLocale`.

For example:

```
GetStarter().clearCachedWindows();
GetStarter().setLocale(java.util.Locale.FRENCH);
```

Note: The `SetLocale` method works under the following limitations:

- Main window menus will not change to the new locale
- The current Subapplication's locale only changes after you have re-entered it
- The setLocale method does not work in clustered applications

Displaying System Messages in the Launcher Applet

```
public void displayMsg (String msg)
```

This method is called when a system message is to be displayed in the launcher applet. Normally a message is localized and passed on for displaying to the super class. You can intercept this method call to process the messages in the original language. Eventually, it is possible to pass the method on, to prevent messages from displaying.

```
public int messageBox(String msgText, String msgTitle,
int msgBoxType, boolean modal)
```

The possible types are:

```
cst.client.ApplWin.CSTInfoDialog.OK_DIALOG
cst.client.ApplWin.CSTInfoDialog.YES_NO_DIALOG
```

This method is called when a message window is to be displayed.

Formatting Text

This feature gives you control over text presentation by creating a Java Class that during runtime formats text from your host application to the GUI window and vice-versa.

For example, a host field containing the text "CA" could be automatically translated to "California" on the GUI window. Date fields, presented on the host as "950211" could be formatted to "11-Feb-95" on the GUI window and typed back to the host in the original format.

With JIS Interface Server, you can write a new Java Class to provide external text-formatting functions. The external Java Class adds on the many text formatting options that come with the product. Creating a Java Class lets you define your own text-formatting functions, suited to the unique aspects of your host application.

This section describes the API functions that you must develop, in order to create a Java Class for text-formatting to and from your GUI window.

Note: This procedure should be performed by Java programmers.

Technical Notes

The following are technical notes:

- The class that you have developed must reside in the class path.
- The class must also be added to the `<ApplName>.ini` file as follows:
  - If it does not already exist, add a section titled [User Defined Format].
  - Add to the section the name of your class as follows:

    ```
    [UserDefinedFormat]
    UseUserFormat=1
    Classes=appls.my_appl.server.user.MyExternalFormat
    UseStandardFormatsDLL=0
    ```
- You should write a Java class that implements the ExternalFormat interface.

ExternalFormat interface Functions

This section describes ExternalFormat interface functions.

Process_text

```
public String process_text(  String function, String textBefore, String
parameter );
```

This function is called by the server whenever the text of a control has to be formatted using an external function. The function performs the string manipulation on the text in the textBefore string, and returns the result. If the function has parameters, they are supplied in the parameter string.

List_of_functions

```
public GALString list_of_functions ();
```

The server uses this function to ask the implemented class for the list of functions it supports. The programmer should fill the return String with the names of the functions separated by a semicolon.

For example: `AlignToLeft;AlignToRight;Add#AtBegining;.....;`

Function_parameters

```
public int function_parameters (String function);
```

The server uses this function to retrieve information about a class-supported function. The method should return 0 if the function has no parameters, else it should return 1.

MyExternalFormat Class Example

```
package appls.<ApplName>.server.user;
import cst.server.basic.*;
import cst.server.utils.formfunc.*;
import cst.server.general.Globals;
public class MyExternalFormat implements ExternalFormat  {
  public String process_text( String function, String
                              textBefore, String parameter ){
    // should return the formatted text according to the input
    // function name.
     return textBefore;
  }
  public int function_parameters (String function){
    // should return 0 if function does not need parameters
    // else return 1.
     return 0;
  }
  public GALString list_of_functions (){
    // should return list of all supported functions separated
    // with semicolons.
     return new GALString("MyFormatFunction;");
  }
  public ExternalFormat InitFormatText (Globals globals){
     return this;
  }
}
```

Extending the Server's Java Code

Client code extensions are useful for performing GUI manipulations, however you sometimes have to write code that is designed to extend the code that runs on the server. Such server code extensions are required to manipulate data on the server or to perform operations that you do not want the client to perform from his computer.

When are Server Extensions Used

Server extensions are used when the client computer cannot perform certain actions, or when you do not want the client computer to perform certain procedures:

- Because of Java Applet Security limitations, the client computer can only access the computer from which the applet was downloaded. If you want the client to access data located on another computer, use a server extension to access that data on behalf of the client.
- If you do not want the client computer to have unrestricted access to a file on the server computer, use a server extension to access the file and send the data to the client computer.
- Use server extensions for performance reasons. For example, when the client needs to search over a large number of records that reside on the server. In certain cases, it may be very inefficient to access all the records from the client machines. The server itself can perform the query much faster and then send results to the client. Another example is for host navigation scripts - in many cases it is much more efficient to complete the navigation on the server and to transfer only the target screen to the client.

Both JIS Interface Server methods and server code extensions run on the server, but there are two reasons to prefer a code extension to an JIS Interface Server method:

- Some Java developers may be more familiar with writing Java code than with writing JIS Interface Server methods.
- JIS Interface Server methods are limited in what they can perform as compared to the Java programming language. For example, DB access (i.e., JDBC) is most efficiently done via the Java language.

Server Java Sources Created During Compilation

The compilation process creates directories and Java source files, amongst which are several that you modify when creating server-side extensions. This section presents a list of such files.

The compilation process creates two directories that contain Java source files that are used by the server:

- `C:\Ace\JacadaFiles\src\appls\<ApplName>\server\original\`
  Contains application data needed by the server.
- `C:\Ace\JacadaFiles\src\appls\<ApplName>\server\user\`
  Contains templates for server code extensions.

> **Note:** Never modify the source files located in the `<InstallDir>\JacadaFiles\src\appls\<ApplName>\server\original` directory, because the compilation process rewrites these files every time it is performed.

To extend your Java server code, only modify the files located in the directory `C:\Ace\JacadaFiles\src\appls\<ApplName>\server\user\`

Files in the server\original Directory

During the compilation process, the JIS Interface Server methods are translated into Java code and stored in several files. Use this translated Java code for writing server extensions. The translated code can be found in the following files:

- `C:\Ace\JacadaFiles\src\appls\<ApplName>\server\original\GeneralInternalSubappl.java`
  Contains the code for the application's General System-Triggered Methods and General User-Triggered Method (GUTM).
- For each Subapplication a file is created, whose name is: `C:\Ace\JacadaFiles\src\appls\<ApplName>\server\original\<SubApplName>.java`
  This file contains the code for the Current Subapplication User-Triggered Method and System-Triggered Methods.

> **Note:** A method's name in the Java code is made up of its name in JIS Interface Server's GUI interface, preceded by a u and an underscore character (u_).

When you write a GUTM called MyMethod, its code is located in `<InstallDir>\JacadaFiles\src\appls\<ApplName>\server\original\GeneralInternalSubappl.java`, and its name in the code is u_MyMethod.

When you modify the Current Subapplication System-Triggered Method UserInitSubApplication in the SUBAPPL Subapplication, its code is located in `C:\Ace\JacadaFiles\src\appls\<ApplName>\server\original\SUBAPPL.java`, and its name in the code is u_UserInitSubApplication.

Server Code Extension Types

Server code extension types are identical to JIS Interface Server method types. You can therefore extend the server's code:

- By overriding a User-Triggered Method that is attached to a trigger. In this case the extension's execution is linked to the trigger's activation by the user during runtime.
- By overriding a System-Triggered Method. In this case the extension's execution is linked to the activation of the System-Triggered Method during runtime.

Creating a Server Code Extension

JIS Interface Server DoMethods can be written either using JIS Interface Server's Method Editor tool or directly using the Server API feature (See "The Server API" on page 247). The great advantage in writing the code yourself lies in the fact that the compilation process can be avoided. However, make sure you are well versed in the Server API feature before you attempt using it. Note also that methods that have controls attached to them MUST be create using JIS Interface Server.

To extend the Java server code:

1. In JIS Interface Server, create the User-Triggered Method (UTM), or modify the System-Triggered Method to be used for the server code extension.
2. If you create a UTM, attach it to a control.
3. Perform Generate Runtime.
4. Write the code extension. For detailed instructions, see the "Writing the Code Extension" on page 246.
5. Use the jacc.bat batch file found in `<InstallDir>\JacadaFiles\src\appls\<ApplName>\server\user\` to compile the Java sources.

> **Note:** If you are writing the methods directly in the user's directory, steps 1-3 are redundant.

Extending a Single Subapplication vs. an Application/Library

The type of JIS Interface Server method you create depends on the application level to which the extension is applied:

- To extend a specific Subapplication, use a Current Subapplication UTM or a Current Subapplication System-Triggered Method.
- To extend an application/library, use a General UTM or a General System-Triggered Method.

Writing the Code Extension

This section details where and how to write your code extension.

To extend a specific Subapplication:

1. In the `server\user\` directory, create a file called `<SubApplName>.java`. The class defined in `\server\user\<SubApplName>.java` must inherit from the class defined in `\server\original\<SubApplName>.java`.
2. Copy the import lines from `\server\original\<SubApplName>.java`.
3. In `\server\original\<SubApplName>.java`, find the code for the method you wish to extend.
4. Copy the code lines into `\server\user\<SubApplName>.java`.
5. Write your extension in the code.
6. Use `C:\Ace\JacadaFiles\src\apls\<SubApplName>\server\user\jacc.bat` to compile the Java sources.

Example 61. Location of code for u_UserInitSubApplication

Suppose you want to extend the System-Triggered Method UserInitSubApplication. Look for the code for `u_UserInitSubApplication` in `<InstallDir>\server\original\<SubApplName>.java`, copy it into `server\user\<SubApplName>.java`. Insert your extension into this code.

> **Note:** If you are writing the methods directly in the user's directory, steps 3-4 are redundant.

To extend an application or a library:

In the file `server\user\GeneralSubapplication.java`
write your extension, overriding the code of the GUTM you created, or of the System-Triggered Method you modified.

> **Note:** In this case there is no need to create a new file since the empty template is automatically generated during the compilation process.

Example 62. Extending GUTMs

Suppose you want to extend the GUTM MyMethod. In the file `server\user\GeneralSubapplication.java`, override the method `U_MyMethod` defined in `server\original\GeneralInternalSubappl.java`.

Code Contents The code extension may contain any valid Java code. The code may use any JDK 1.1 API or JDK 1.2 API when running the server using JDK 1.2 JVM. However, JIS Interface Server provides you with a fixed API that corresponds to the full set of JIS Interface Server's DoMethods.

The Server API

The Server API feature is a complementary utility for server-side code extensions. Java code extensions can use this API to interact with JIS's internal data structures. Using the Server API enables you for example to

- Set the text of a field.
- Read from the INI file (see example).

The Server API Interfaces

The Server API consists of a multitude of interfaces. These different API server Interfaces are included within the two sub-packages of the cst.server.export.api package.

The two sub-packages are:

- `cst.server.export.api.window` — includes classes that implement all the different controls.
- `cst.server.export.api.general` — includes classes that implement all other objects. In addition, this package includes the JSString class that takes care of all character related issues. This class inherits from the cst.server.basic package.

The Server API

This section provides a detailed look at the Server API.

cst.server.export.api package

Table 42. cst.server.export.api package (Sheet 1 of 2)

cst.server.export.api.general	cst.server.export.api.window
ISystem	IWindow
IApplication	IDubWindow
ISubApplication	IListBox
IScreen	IComboBox
IDBSession	IEdit
IExternalData	IAdjustableEdit
IGeneral	IStatic
IJcsSession	IGroupBox
	IPushButton
	ICheckBox
	IRadioGroup
	ITable
	ITabs
	IPictureButton
	IDate
	IPrompt
	IFrame

Table 42. cst.server.export.api package (Sheet 2 of 2)

cst.server.export.api.general	cst.server.export.api.window
	IGUIObject

JIS's javadoc Files

JIS provides you with a set of HTML files that include a full listing of the server API interfaces, the methods implementing them, and explanations about how to work with them.

The javadoc files are installed under the following directory:

`<InstallDir>\JacadaFiles\docs\server`

Example 63. JIS's javadoc file location

<InstallDir>\JacadaFiles\docs\server

Example of How to Write a Java Server Extension

In the following example, we show you how to write a Java server extension for an application called `MYAPPL`. The extension is for a specific Subapplication called `SIGNON`.

The code will extend the functionality of the `u_UserInitSubApplication( )` method, by changing a value read from the INI file to upper case.

1. Create a new file in `C:\Ace\JacadaFiles\src\appls\MYAPPL\server\user` called SIGNON.java.
2. In the new file, create a class called `SIGNON` (in the package `appls.MYAPPL.server.user`) that extends `appls.MYAPPL.server.original.SIGNON`
3. Open the file `C:\Ace\JacadaFiles\src\appls\MYAPPL\server\original\SIGNON.java`.
4. Into the new file, copy the import lines from the original file.

5. In the new file, write the code that extends the `u_UserInitSubApplication ( )` method. The code extension reads a value from the INI file and changes this value to upper case.
6. Save the new file.
7. Use `C:\Ace\JacadaFiles\src\appls\MYAPPL\server\user\jacc.bat` to compile the Java sources.

This is the source code you finally obtain in the new file:

```
import cst.server.api.*;
import cst.server.applicat.convertr.runtime.*;
import cst.server.applicat.convertr.cvrt.*;
import cst.server.applicat.*;
import cst.server.window.*;
import cst.server.basic.*;
import cst.server.collect.*;
import cst.server.struct.*;
import cst.server.general.*;
import cst.server.gds.*;
import cst.util.*;
import cst.server.export.api.window.*;
import cst.server.export.api.general.*;

package appls.MYAPPL.server.user;
public class SIGNON extends appls.MYAPPL.server.original.SIGNON {
   public Object u_UserInitSubApplication ( ) {
      String rc_0 = null;
      // Read profile from ini file and change value to
          // uppercase:
   rc_0 = getSystem().getApplPrivateProfileString
                                    ("profile", "user" ,"" );
   rc_0 = rc_0.toUpperCase();

   System.out.println("Testing: " + rc_0);
   // Set the text of an edit:
   ((IAdjustableEdit)getControl("User")).setText(rc_0);
   return this;
   }
}
```

Summary: Client vs. Server Code Extensions

The table below summarizes the different files and operations that are involved in the process of writing a code extension, depending on whether you extend client or server code.

For lack of space, the entire path to a file is not indicated.

Insert `C:\Ace\JacadaFiles\src\appls\<ApplName>` at the beginning of the path when extending the main application's code or the code of a Subapplication that belongs to the main application.

Insert `<InstallDir>\JacadaFiles\src\appls\<LibName>` at the beginning of the path when extending a library's code or the code of a Subapplication that belongs to a library.

Table 43. Client and server side extensions types and usage

	Client-side extension	Server-side extension
The extension applies to the Application/ Library level	Modify the file: `\user\ApplSubapplWindow.java`	Modify the file: `\server\user\GeneralSubapplication.java`
The extension applies to a Specific Subapplication	Create the file: `\user\<SubApplName>.java` which inherits from `\original\<SubApplName>.java` and write your extension. Register extension class in `\user\JacadaStarter.java` by calling `addWindow ()` (See "Extending the Code of a Subapplication" on page 198).	Create the file: `\server\user\<SubApplName>.java` which inherits from `\server\original\<SubApplName>.java` and write your extension. **Note:** there is no need to Register the extension class.
Compile the code extension using:	`\user\jacc.bat`	`\server\user\jacc.bat`

Appendix A. Java Client Limitations

This section discusses limitations in the Java Client and special considerations you must be aware of when developing a Java Client application in ACE. The following tables list Controls, Control Attributes, Features and Fonts which are either not supported or only partially supported in this version.

Partially Supported ACE Controls

The following controls are not supported or only partially supported in this version. Control Attributes lists the control's unsupported attributes. Under Comments you will find the limitation of the control or of the control attribute.

Table 44. Partially supported ACE controls (Sheet 1 of 4)

Control	Control Attribute	Comments
Buttons	Ampersand (&) support	
		Splitting text into two lines using the ^ character is not supported
Combo	OEM convert Auto horizontal scroll Integral height	
Edit	Hide selection Auto horizontal scroll	Default behavior

Table 44. Partially supported ACE controls (Sheet 2 of 4)

Control	Control Attribute	Comments
Multiline Edit	Lowercase doesn't work in Appletviewer RMB doesn't work Cut/copy/paste/undo Horizontal scrollbar is permanent. The settings in Ace have no effect.	
Dynamic Area		All dynamic controls are supported, though their size might be different than shown in ACE.
Frame	Hide selection	Hiding the Frame control also hides everything inside it.
Spin	Hide selection Auto horizontal scroll	Default behavior
Static	Style type - Wrap	
Prompt	Hide Selection	
Menus	Ampersand (&) support for menu items	
Numeric field		Color indicators are ignored.
Editing commands (Cut/Copy/Paste/ Undo)		Can only be performed using signed applets. See "Signed and Unsigned Archive Files" on page 61.

Table 44. Partially supported ACE controls (Sheet 3 of 4)

Control	Control Attribute	Comments
Table Control	Methods that are activated when a list is in focus or when a selection has changed Three state check box The text inside controls such as edits is aligned to the top, and not the center as in ACE. This is apparent when you increase the size of the table's rows Masks for Controls Headers	Splitting text into two lines using the ^ character is not supported. Refrain from changing the default splitting mechanism.
Tabs		Tabs are aligned on one horizontal line. When you have many tabs, their labels may be reduced to small font sizes.
	Folder caption	Splitting text into two lines using the ^ character is not supported
Popup Subwindow		Not supported
Popup Windows		When the main window is in focus the popup Subapplication disappears behind it. This limitation can be overridden by setting the “UseModalPopups” html parameter to TRUE.

Table 44. Partially supported ACE controls (Sheet 4 of 4)

Control	Control Attribute	Comments
	Modal Popups	The popup window has no menus. The main window's menus are not accessible. This means that the user cannot perform such operations as exiting the application, opening the host window, etc. while the popup is active. Message boxes may disappear *behind* the popup, if the user clicks the popup window while the message box is open. When closing a calendar window that has been opened from a date control inside the popup, the focus jumps back to the main window. In some browsers, closing a modal dialog causes the screen to flicker.

Note: If a Subapplication uses a control that is not supported, the compilation process generates an error message and terminates.

Partially Supported Features

The following feature is partially supported in this version

Table 45. Partially supported features

Feature	Supported Aspects of the Feature
MDI	• Overlapping windows • Child. When maximized, the child's caption is added to the main window's caption

Supported Fonts

The following fonts are supported. If a Subapplication uses an unsupported font, Generate Runtime generates a warning message and continues working, using a different font. Note that Windows default fonts may look differently when displayed in Java. We therefore recommend that you choose the following fonts, supported by the Java Client:

Table 46. Supported fonts

Font Selected in ACE	Font Displayed in the Java Runtime
MS Sans Serif	Dialog
Times New Roman	Times Roman
Arial	Helvetica
Courier New	Courier

The font size displayed in the Java runtime will automatically increase by approximately 33% in order to match the original size selected in ACE

Table 47. ACE versus Java font sizes

ACE Font Size	Java Font Size
8	11
9	12
10	13
11	15
12	16

Color Support

System colors—fixed system colors as well as system colors chosen from a color table—are supported. Note, however, the following remarks:

- Color behavior may differ according to the browser running the application. To minimize the difference we recommend you use a graphic system that allows more than 256 colors.
- A difference in color presentation may occur since Java's system colors are not completely identical to Windows system colors.
- The 'Scrollbar' system color does not work when used in a color table.

Unpacking an Application Without its Libraries

Unpacking an application without all or part of its libraries (or unpacking an application that has been packed without its libraries) requires special attention in two cases:

- When unpacking to a new environment.
- When unpacking to an environment that contains the main application and not the associated libraries.

In both cases, once you unpack you must modify the library list that is located in:

- The `<ApplName>.ini` file.
  The library list is located under the [Program] section, in the "Libraries=" setting.
- The runtime environment window.
  The library list is located in the General Options dialog box, under the Navigation tab.

In both locations, you must delete the libraries that have not been unpacked.

If you neglect to do so, the JIS Interface Server runtime will not work since it points to nonexistent libraries.

JIS Server Method Limitations

The following limitations apply to JIS Server methods:

- Some methods that handle the window's runtime behavior are not supported.
- Subapplication-specific methods do not work on the toolbar.
- The HasFocus and FocusedControlName methods are responsible for identifying which control is in focus and reporting it. They correctly identify and report a Table that is in focus, but not the exact control inside the Table.

- Changes made to default RMB methods written in ACE are not supported by the Java Client. RMB functionality is obtained through extension of the Java code.
- When you run an application without a user profile, and the application calls a DoMethod that writes information to the INI file, this information will be effective for the current sessions only, and will not be saved to file.

Restrictions on JIS Server Supported Methods

All methods created in the converter are automatically translated into Java during the compilation process.

Generating JIS Server Methods During Compilation When you create a new method or update an existing one, make sure that the variable values are correct and in accordance with other variables within the method's syntax. This is important since the Java compilation process requires data about method return types and variables types. When during the compilation process a contradiction is detected within a method's syntax the method is not compiled.

Using the error messages issued by Generate Runtime and the Java compiler, you must correct the syntax errors. This section provides you with a list of System-Triggered Methods included with ACE and the Generate Runtime check-list upon compiling a method for the JIS Server.

An error in a method's syntax can be detected either:

- During Generate Runtime. Generate Runtime issues an error message describing the nature of the problem.
- During the Java compilation. In the Java compiler prompt, error lines indicate the error type and the location in the compiled Java file where the error was detected.

Following is the Generate Runtime check-list upon compiling a method for the JIS Server:

- Checks the data type of all the variables

  Example of valid syntax:

  ```
  #0 = DoMethod: Receiver: 'Application' Method: IsJavaServer Parms ( )
  ```
- The variable is assigned a value: The IsJavaServer DoMethod returns a Boolean value.

  Example of invalid syntax:

  ```
  #1 = DoMethod: Receiver: 'System' Method: Beep ( )
  ```
- A variable has to be assigned a value. Since the Beep DoMethod returns 'void' (i.e. does not return a value), no value will be assigned to the variable.
- Check for consistency in the values returned by the method

  Example of invalid syntax:

  ```
  #0 = DoMethod: Receiver: 'Screen' Method: IsHostPopup Parms ( )
  ```

```
If: Cond: '#0'
Return = False
Else:
Return = Fail
EndIf:
```

- The first return is of Boolean type and the second Null. This is wrong since a method can not return more than one data type.

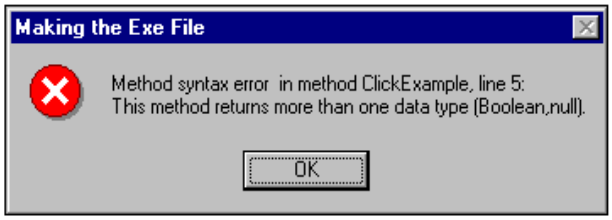


Figure 36. Method syntax error

- Checks the validity of IF and MSGBOX statements
- A condition of IF must be Boolean
- The Caption and Message of MSGBOX must be String

  Example of invalid syntax:

```
#0 = DoMethod: Receiver: `SubApplication` Method: Name Parms: (  )
#1 = DoMethod: Receiver: `#0` Method: FindSubSet Parms: ( `0` , `"ABC"`  )
If: Cond: `#1 !=  _FAIL `
    Return: NoValue
EndIf:
```

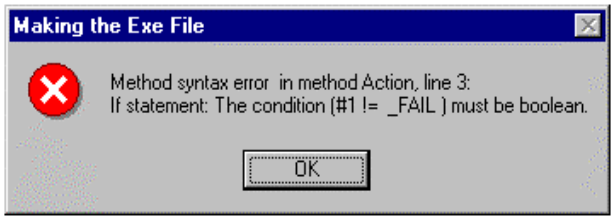


Figure 37. Method syntax error 2

Note: The JIS Server does not support the FindFrom and FindSubset DoMethods. Instead you can use the corresponding FindFromInt and FindSubsetInt DoMethods.

- Checks that the data type of parameters that are passed to DoMethods is correct

  Example of valid syntax:

```
#0 = Expression: Expr: 'TRUE'
```

```
DoMethod: Receiver: 'Screen' Method: TypeStringWithoutCheck: ('#0')
```

This is correct since the TypeStringWithoutCheck DoMethod expects and receives a Boolean parameter.

Example of invalid syntax:

```
#0 = Expression: Expr: 'TRUE'
DoMethod: Receiver: 'Window' Method: RunHelpFileByIndex Parms: ('#0')
```

This is wrong since the RunHelpFileByIndex DoMethod expects a String parameter but receives a Boolean parameter instead.

Note: The only line types that support returning values are **DoMethod**, **Expression** and **MsgBox**.

System-Triggered Methods Included with ACE

ACE includes a number of hard-coded methods that you can customize to your application. Note that the methods you customize must return the appropriate values, as follows:

Table 48. System-Triggered Methods included with ACE (Sheet 1 of 2)

Method Name	Return Value
UserIsRealMessage	TRUE / FALSE
UserInitSubApplication	SUCCESS / FAIL
UserInitBeforeFirstSubAppl	TRUE / FALSE
UserAfterRefreshSubApplication	TRUE / FALSE
UserRefreshSubApplication	TRUE / FALSE
UserShouldCloseSubApplWindow	TRUE / FALSE
UserCloseSubApplWindow	TRUE / FALSE
UserDestroySubApplication	SUCCESS / FAIL
TableChangedSelection	SUCCESS / FAIL
PageDown	SUCCESS / FAIL

Table 48. System-Triggered Methods included with ACE (Sheet 2 of 2)

Method Name	Return Value
PageUp	SUCCESS / FAIL
GetToTopOfList	TRUE / FALSE
GetToBottomOfList	TRUE / FALSE

Note: UserRMB methods are not supported. You can obtain Right Mouse Button functionality by extending the Java code.

Appendix B. Troubleshooting

This section describes some of the problems that you may encounter and some solutions that may help you when working with the Java Client.

Table 49. Troubleshooting and solutions (Sheet 1 of 4)

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. Communication problems. You are behind a firewall	Activate the JIS Server. 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 validate that communication exists between the client and the server. In order to run the JIS-delivered applications you will need to use HTTP communication, or enable communication through the Ports that JIS Interface Server uses. The Ports are 1100 and 1101.
Java code may be out of date.	Java client and JIS Server were not produced during the same runtime generation.	Generate runtime again. Clear your Browser's disk cache.
Application xxx is not installed on the Server. Try another Application.	Application was not found in the jacadasv.ini file. The `<ApplName>.ini` file is missing	Install the application on the Server computer Recompile the Application to generate the `<ApplName>.ini` file.

Table 49. Troubleshooting and solutions (Sheet 2 of 4)

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 GmbH 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.
Application xxx is not properly installed.	The Application was added to the JIS *.ini file, but was not specified correctly.	Try to activate the Application on the JIS Server Computer in the Java Client runtime. Re-install the Application.
Generate runtime fails and you get the following message: *Error: Unable to delete the file:* `c://JacadaFiles/classes/appls/<ApplName>...cvrecord.dir`	Generate runtime cannot process 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.

Table 49. Troubleshooting and solutions (Sheet 3 of 4)

Problem / Message	Possible Reasons	Solution/s
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.
Sluggish response when using Host View intensively with an HTTPS connection.	The client may be generating more traffic than your network connection can comfortably handle.	In the application's applet (the file named `<ApplName>.html`), add the parameter `SendEmulatorKeyTypedEvent` and set its value to "false", like so: `<PARAM name = "SendEmulatorKeyTypedEvent" value = "true">`. This parameter setting instructs the applet to send one key event (pressed) instead of sending two events (pressed and typed) for each interaction. The result is less network traffic and improved response time.

Table 49. Troubleshooting and solutions (Sheet 4 of 4)

Problem / Message	Possible Reasons	Solution/s
Client connection to the JIS Server is up to two minutes when run from another PC on the network.	Erroneous network configuration of the relevant PCs. The problem is usually in the DNS configuration.	**Solution 1** **1** Fix the DNS configuration of the Client computer and Server computer: **2** From Control Panel > Network > TCP/IP > Properties, access the DNS Configuration tab. **3** Verify that the DNS Host contains the DNS IP address and that the Domain is correct. **4** If the computers are not in the same subnet, verify also that the gateway setting is correct in Control Panel > Network > TCP/IP > Properties > Gateway. **Solution 2** If there is no DNS, provide each PC with a table that contains the Host name's translation to the IP addresses. In each PC: **1** Under the Win95 directory, create a file called Hosts (look at hosts.sam for an example). **2** Enter both computers' names and addresses. **3** Restart the PC for this change to take effect. **4** From a DOS prompt, run "ping`<OtherHostName>`" to verify they recognize each other. (`<OtherHostName>` should be replaced with the actual name of the other PC).

Appendix C. Directory and File Structures

This section details the directory structure of the development environment. Use these pages as a reference when searching for specific files.

Development Environment Directory Trees

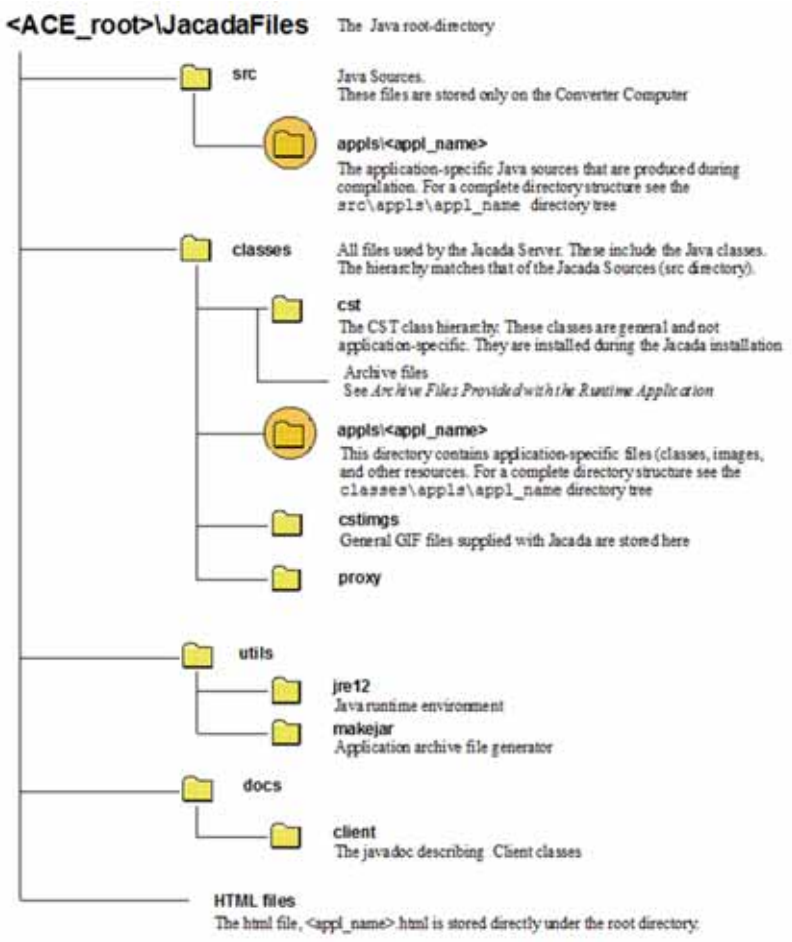


Figure 38. Java root directory file structure

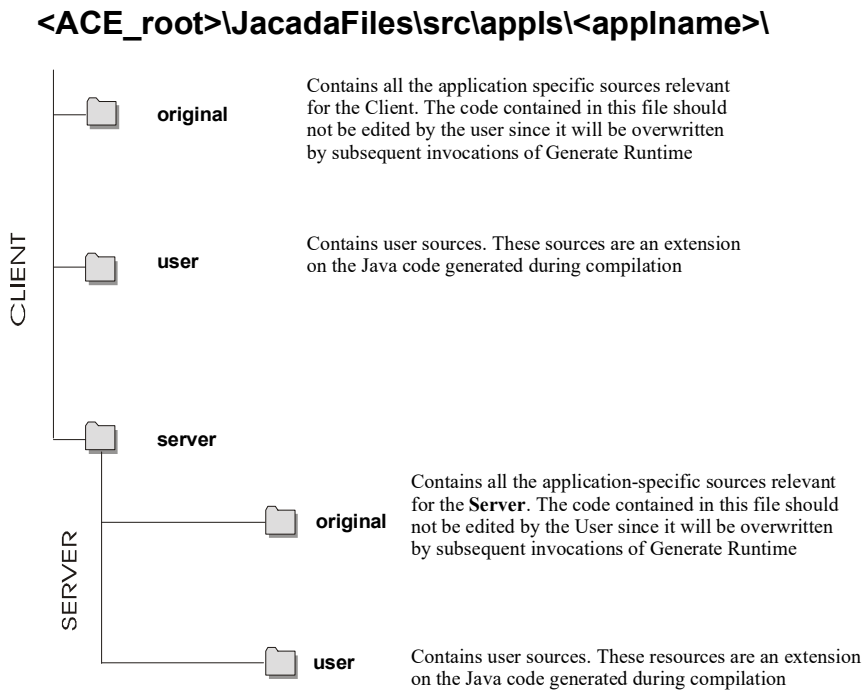


Figure 39. Client and Server Java source file directory structure

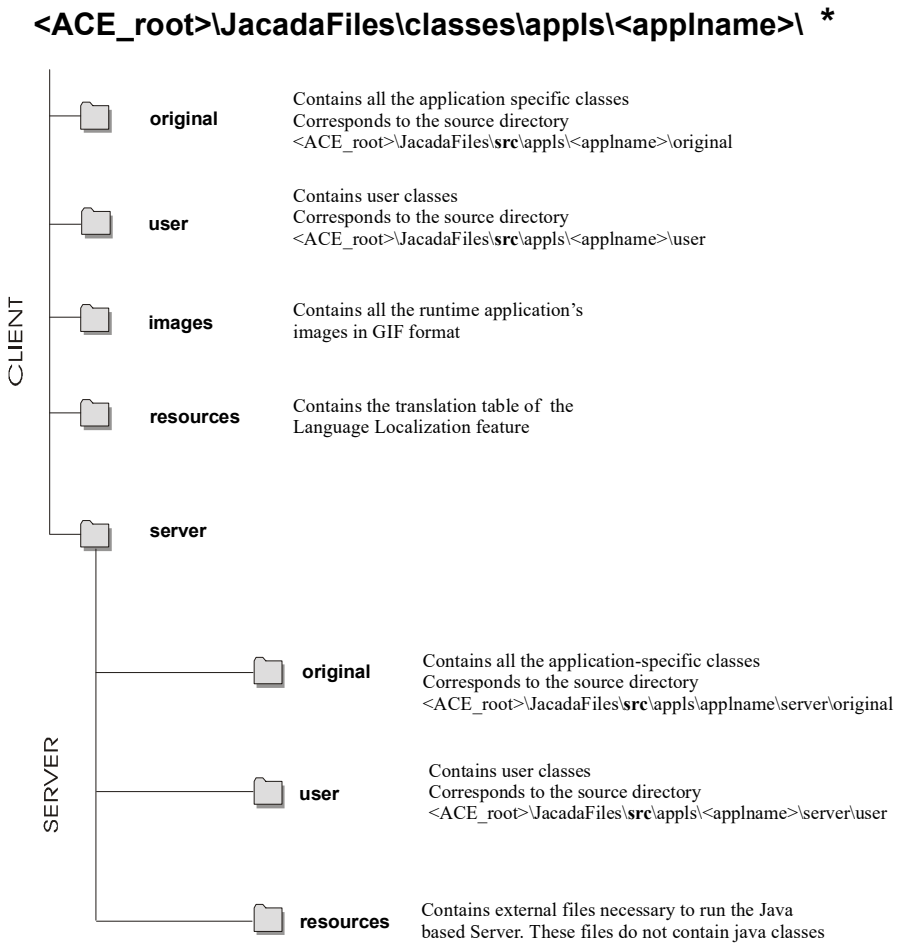


Figure 40. Client and Server Java class file directory structure

> **Note:** After performing a runtime installation, the classes are placed under the runtime root directory,

Example 64. Placement of classes after runtime generation

```
<RuntimeRootDir>\classes\appls\<ApplName>.
```

The Users Runtime Directory Tree

In the following illustration:

- The application Sales is installed on the server.
- There are two users: Brad and Janet.

Note: You must create manually the USERS directory and all the directories and files it contains.

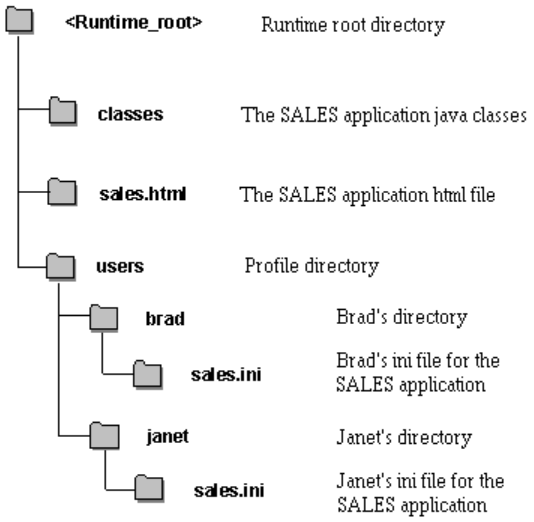


Figure 41. Users runtime directory structure

Appendix D. Glossary of Terms

The following is a Glossary of terms which will assist you in familiarizing yourself with the JIS Interface Server's Java Client.

Table 50. Glossary of terms (Sheet 1 of 2)

Term	Definition
Java Source File	A file with the extension ***.Java**, contains Java source code.
Java Class File	A file with the extension ***.class**, contains a compiled Java code. The Java compiler produces ***.class** files from the ***.Java** sources.
Java Applet	A program, written in Java, designed to be **run** by a Web Browser. It consists of one or more Java classes. Applets are Java programs that are downloaded over the WWW and executed by a Web Browser on the user's computer. Applets depend on a Java-enabled Browser in order to run.
HTML File	A file written in HTML (Hyper-Text Markup Language), can be displayed by a Web Browser.
Web Browser	A Program that supports browsing HTML files. A Java-enabled Web Browser is a Web Browser capable of executing Java applets.
Server Computer	A computer that runs a Web Server Program and the JIS Server Program. This is the computer that users contact from their Web Browser.
Web Server Program	A Server Program, running on the Server Computer, that responds to HTTP requests from Web Browsers, sending files according to their requests.

Table 50. Glossary of terms (Sheet 2 of 2)

Term	Definition
JIS Server	A Server Program running on the Server Computer. It opens a session on the host (mainframe or iSeries) from one side, and interacts with the Client on the other side. The interaction with the Client is done by communicating with a Java applet running on the user's computer.

Index

A

activate (JacadaStarter method) 233
Adding a Background Image 204
Adding Action Buttons to a Subapplication 205
Adding Action Buttons to the Main Window Tool Bar 207
Adding Animated Buttons to Subapplications 210
Adding Bubble Help to Components 208
Adding Content to a Table Cell 212
AddWindow 227
AllowSaveHostScreenImage 51
AnimatedLogoDelay 50
AnimatedLogoImages 50
AnimatedLogoVisible 50
Applet
  running Jacada using a launcher 63
Application_Name
  ...see Jacada INI file settings
ApplicationIconLarge 52
ApplicationIconSmall 52
Archive files 59
  reducing download time 67

B

Button control
  methods for querying the button control 207
ButtonCursorType 51
ButtonRolloverBackground 50
ButtonRolloverForeground 51

C

Changing the background color of the printed window 51, 190
Changing the Titles of Tab Control Folders in Runtime 219
Classpath
  ...see Jacada INI file settings
ClientConnectionMessage 57
ClientTerminationMessage 57
ComboFitHeight 49
ComboFitWidth 49
Command line parameters 42
CommTimeOut 53
Compilation
  executing 24
  setting options 22
ConnectionFailed 56
ConnectionRetryInterval 55
ConnectionRetryTimeout 55
ConnectPort 54
ConnectPortURL 54
Controlling the scale of the printed window 190
Controlling the scale of the printout 51
Converting images 25
Creating Controls 200
Creating custom validity checks 221
Creating Logic Peers 201

D

Data sharing between client and server 225
Debug filters 110 to 112
  defining in the jacadasv.ini file 76
Debug tab 136
Debugging the Jacada Administrator 129
DebugLevel 55
DebugTimeStamp 56
DefaultDateFormat 49
Defining Number and Length of Lines in Multi-line Edits 211
Delegation Event Model 201
Displaying message boxes 219
Dump Files 112

E

Enable (SessionCountLog)
  ...see Jacada INI file settings
Enable (SessionLog)
  ...see Jacada INI file settings
Enable (XMLLog)
  ...see Jacada INI file settings
Enable (XMLServer)

...see Jacada INI file settings
Enabling the list menu 213
Event Handling 201
ExitPage 56
Extending the Java Code
adding a background image 204
adding Action Buttons to a Subapplication 205
adding Action Buttons to Main Window toolbar 207
adding animated buttons to subapplications 210
adding bubble help to components 208
Adding Content to a Table Cell 212
addWindow 227
Changing the Titles of Tab Control Folders in Runtime 219
creating controls 200
creating custom validity checks 221
creating logic peers 201
data sharing between client and server 225
defining number and length of lines in multi-line edits 211
deprecated methods 197
displaying message boxes 219
Enabling the list menu 213
event handling 201
focus and tabbing management 203
Handling table selection events 213
keyboard management 203
manipulating host originated data 214
modifying the default floating menus behavior 217
multi-character search in combo boxes 224
of all subapplications 200
of one subapplication 198
of the Main Window 199
RMB floating menus support 216
server-side extensions 243
updating menu items in runtime 211
ExternalFormat class 241

F

File (SessionCountLog)
...see Jacada INI file settings
File (SessionLog)
...see Jacada INI file settings
File (XMLLog)
...see Jacada INI file settings
FileInterval
...see Jacada INI file settings
Focus and Tabbing Management 203
Font substitution
about 69
debugging 71
resource file 69
Formatting text 240
FullClientFrame 58
FullClientURL 58
Function_parameters 242

G

Generating a runtime
... see Compilation
getCurrentPanel (JacadaStarter method) 233
GetTextFromUserTimeout
...see Jacada INI file settings
GUIPrintingBackground 51
GUIPrintingInMonochrome 51
GUIPrintingScale 51

H

Handling table selection events 213
HideApplicationMenu 49
hideMainWindow (JacadaStarter method) 234
HideMenus 48
HideToolbar 48
HostScreenDisplaySize 50
HTML Settings 48
HttpAddr 54
HTTPPortRange
...see Jacada INI file settings
HttpReuseConnection 55

I

Image conversion 25
INI File Settings
...see JIS INI file settings
IniDir
...see Jacada INI file settings
IniVersion
...see Jacada INI file settings
Installing multiple applications on the same Jacada Server 41

J

Jacada Administator
  Debug tab 136
Jacada Administrator 127
  interface 130
  starting 127
Jacada INI File Settings ?? to 88
Jacada launchers
  customizing 230
  JacadaBasicLauncher 63
  JacadaLoginLauncher 63
Jacada Runtime Directories Tree 270
Jacada Server
  activating from an NT 31
  activating from Linux 40
  command line parameters 42
  Jacada Administrator 127
  optimizing 73
  running as a Windows service 147
  scaleable system structure 89
JacadaStarter
  code examples 235
  methods
    activate 233
    addWindow 227
    getCurrentPanel 233
    hideMainWindow 234
    waitForFirstWindow 232
    waitForNextWindow 235
Jacadasv.ini 74
jacadasv.ini 96
  for scaleable server system 96
  general structure 96
  setting logclasses 108
jacservr.exe 31
JAM
  ... see Jacada Administrator
Java
  documentation for generated classes 196
Java .class Files 196
Java Application
  running the Java Client 64
Java Class File, definition 271
Java Classes
  reducing download time using archives 67
Java Code
  class files 193
  extending 196
  sources within CST hierarchy 193
  sources within User hierarchy 193
Java code extension
  server 243
Java sources of the server 243
Java Sources, compiling 195
Javadoc 196
JavaMemory
  ...see Jacada INI file settings
JavaOptions
  ...see Jacada INI file settings
JavaVM
  ...see Jacada INI file settings
JBSService.exe 149
JBSToService.exe 147
JIS INI File Settings 74 to ??

K

KeepAlive
  ...see Jacada INI file settings
KeepAliveTimeout
  ...see Jacada INI file settings
KeepAliveTimerTick
  ...see Jacada INI file settings
Keyboard Management 203

L

language localization
  activating 154
  current limitations 163
  debugging your localized application 160
  resource files 155
  resource maintenance 157
  setting the mechanism
    through the HTML file 158
    through the JacadaStarter API 158
  string types handled by 159
  work flow 154
LanguageNotSupported 56
Launching Jacada from an Applet 228
Limitations
  Jacada Server methods 259
  Java Client 253
LInux
  installing the runtime on 34
Linux

activating the Jacada Server 40
mapping the PC to 34
List_of_functions 241
Load Balancing 91
Locale_Country 53
Locale_Language 53
Locale_Variant 53
LocaleDebugMode 55
Localization API
implementing the localization feature 239
LogClasses
...see Jacada INI file settings
Logging support
acrchitecture 104
components 104
log classes 106
parameters 106

M

MachineApplications
...see Jacada INI file settings
MachineSessions
...see Jacada INI file settings
Main Window
extending the code of 199
Mainframe
mapping the PC to 34
makejar.bat 68
Manipulating host originated data 214
MaxMachineApplications
...see Jacada INI file settings
MaxMachineSessions
...see Jacada INI file settings
MaxProcessApplications
...see Jacada INI file settings
MaxProcesses
...see Jacada INI file settings
MaxProcessSessions
...see Jacada INI file settings
Modifying the default floating menus behavior 217
MsgboxTimeout
...see Jacada INI file settings
multi-character search in combo boxes 224
MultiMachines
...see Jacada INI file settings
Multiple applications
installing on the same Jacada Server 41
Multiple Server-Computer System 92

O

Optimizing Fonts for Non-Windows Platforms 69

P

PanelTimeout
...see Jacada INI file settings
PortScanRetries
...see Jacada INI file settings
Printer Emulation
adding an HTML parameter to read the client classes 183
adding an HTML parameter to read the printer emulation archive 169
API architecture 177
creating a printer emulation instance 178
establishing print parameters in the runtime ini file 167, 181
examples of how to use the extended printer emulation 185
Host-to-Client printing architecture 165
initializing a default printer emulation session 170
initializing an extended printer emulation session 180
making printing operational 166
providing security permissions 184
sending the print stream to the client 184
tracing printer emulation problems 170
troubleshooting 176
User Interface 174
using 171
Printing the Client host screen 191
Printing the client window
activating 187
changing the background color of the printed window 190
controlling the scale of the printout 190
granting permission 189
modifying through code extension 188
Page setup 172
Process_text 241
ProcessApplications

...see Jacada INI file settings
ProcessSessions
...see Jacada INI file settings
Profile 57
Profiles 150

R

RecvTimeout
...see Jacada INI file settings
RegistryPortRange
...see Jacada INI file settings
RegistrySpawnTimeout
...see Jacada INI file settings
ReportsToMachine
...see Jacada INI file settings
RequestProfile 57
ResourceBase
...see Jacada INI file settings
RMB floating menus support 216
RMISocketTimeout
...see Jacada INI file settings
RtRootDir
...see Jacada INI file settings
RunInsideBrowser 52
Running the Jacada Client inside a Browser Window 66
Running the Java Client from a Java application 64
Running the Java Client from an Applet using a launcher 63
Runtime Application
activating on the Client 44
work flow 29
Runtime Installation
creating 26
installing 26
Runtime menus
updating in runtime 211
Runtime termination 112

S

Scalability 89
client-host connection 93
jacadasv.ini 96
load balancing 91
multiple server computer system 92
single server computer system 90
Server 54
Server Configuration Checker 120
Offline Mode 123
Server Mode 123
server Java code extension 243
server Java sources 243
ServerPortRange
...see Jacada INI file settings
ServerPortRange2
...see Jacada INI file settings
Service
running the Jacada Server as a 147
SessionCountLog
...see Jacada INI file settings
SessionLog
...see Jacada INI file settings
Sharing variables 225
Single server computer system 90
SocketImplFactory
...see Jacada INI file settings
SoftLimitMarginPercent
...see Jacada INI file settings
SpareSessionsPercent
...see Jacada INI file settings
SpawnInterval
...see Jacada INI file settings
StartScanAtRandomPort
...see Jacada INI file settings
StartUpSessionsPercent
...see Jacada INI file settings
StdoutEncoding
...see Jacada INI file settings
Subapplications
extending the code of all subapplication 200
extending the code of one subapplication 198
SystemConnectionTimeout
...see Jacada INI file settings

T

Terms 271
TimerTick (SessionLog)
...see Jacada INI file settings
TimerTick (XMLLog)
...see Jacada INI file settings
TimerTick (XMLServer)
...see Jacada INI file settings

Troubleshooting 263

U

Unpacking an application
limitation 258
UseHttp 54
UseModalPopups 49
UseMultirowTabFolders 52
UseNewHTML 58
UsePorts 54
User Files
overwriting 195

V

Validity checks 221
Variables sharing 225
VersionMismatchPage 56

W

waitForFirstWindow (JacadaStarter method) 232
waitForNextWindow (JacadaStarter method) 235
WaitForSpawned
...see Jacada INI file settings
windowDataReady 214
WindowHScrollIncrement 52
windowReadyForAction 214
Windows service
running the Jacada Server as 147
WindowScrollRepeatInterval 52
WindowVScrollIncrement 52
WISE 30, 32
Work flow 29
WorkingDirectory
...see Jacada INI file settings

X

XMLLog
...see Jacada INI file settings
XMLServer
..see Jacada INI file settings