Product: JIS
Released January 2012
Release Notes for Version 9.1.2

Contents of Version 9.1.2

Contents of Version 9.1.2 1
Installation & Upgrade Information 4
Supported Platforms 4
Recommended Configurations 4
New Features in Version 9.1.2 7
Creating screen images from Natural Maps 7
Simplified HTTPS/SSL Configuration 12
IPv6 Support 15
Specifying a Folder where the Java Client Log File will be Saved 16
Logging Messages Improvements 16
Proxy Servlet Improvements 16
Updated JIS Perl to Version 5.12.2.0 17
Session Dump Improvements 17
Access Log 17
Pattern Matching according to Character Attributes 18
Detailed Description of Version 9.1.2 Fixes 20
Installation 20
JAVA Client 20
XHTML Client 21
Server 22
Innovator 22
Limitations 23
New Features in Version 9.1.1 24
XHTML 24
Server Changes 24
Monitoring Improvements 24
Localization Improvements 27
ACE 27
Runtime Installation 27
Java Client Improvements 28
New Features in Version 9.1 28
WebSphere 7 Support 28
Usability Improvements in the "Generate Runtime" and "Run Application" Wizards 29
Restarting the JIS Server 31

JMX Support ... 32
XHTML Page Size Optimization Improvements ... 34
Server Log ... 35
Java Client Log ... 36
XHTML JavaScript Client Log ... 36
Keyboard shortcut for Java client Print GUI ... 37
Localization Improvements ... 37
Localizing Dynamic Control Strings ... 37
Multiplying Default Control Size by a Pre-Defined Factor ... 39

New Features in Service Pack 9.0.4 ... 41

Logging Improvements ... 41
Simplifying JIS Windows Service Configuration ... 42
Using JAM as an Applet in JIS Standalone Server ... 42
JIS Administrator Command Line Operations ... 43
Modifications made to the J2EE Deployment Procedure (XHTML only) ... 44
Upgrade to Jetty 6.1 ... 45
Running the JacadaProxyServlet as part of the JIS Server ... 46
Reduction of the size of the XHTML file. ... 47
Allowing the User to Adjust the Java Client Debug Level ... 47
Post Class Path ... 47
New Methods for Handling User Variables ... 47
Rebranding ... 49

New Features in Service Pack 9.0.3 ... 49

Changes in the Product Name ... 49
Runtime Installation Improvements ... 49
Simplifying the Printing Emulation Configuration (XHTML) ... 51
Improved Host Language Support ... 51
Java Client "About" Dialog Box ... 57
"Host Print Transform" Printing using Java Services ... 57
Exposing the XHTML Page DOM for Java Extensions ... 57
AutoSkip Supported in XHTML ... 60

New Features in Service Pack 9.0B ... 60

Command-Line Access to ACE ... 60
Improved User Interface ... 64
Additional Enhancements ... 66

New Features in Service Pack 9.0A07 ... 69

Refreshing the XHTML Client When a Page on the Host is Updated ... 69

New Features in Service Pack 9.0A06 ... 70

Support for Keyboard Buffering ... 70

New Features in Service Pack 9.0A05 ... 71

API available to trigger server methods ... 71

Printing 71

New Features in Service Pack 9.0A02 72

Enabling reconnecting to a database after the connection or session fails 72

Limiting the size of the server log files 73

New Features in Service Pack 9.0A01 74

Enable opening a window in a maximized state 74

Deploying a service to a J2EE Server 74

New Features in Service Pack 9.0A00 75

XINIT keyword in BMS maps now supported 75

Maximum permitted size of ACE method increased 75

Print setup dialog can be skipped 75

New methods for setting colors of selected cells 76

Installation & Upgrade Information

Service pack JIS 9.1.2 is focused on adding support for creating screen images from Natural maps and simplifying the configuration of secured communication using Https and SSL as well as implementing various incremental improvements to the product.

Supported Platforms

Windows Server 2003 Standard and Enterprise Edition (32-bit)

Windows Server 2008 Standard and Enterprise (32-bit)

Windows Server 2008 Standard and Enterprise (64-bit)

Windows XP Professional (32-bit)

Windows Vista (32-bit)

Windows 7 Professional, Ultimate and Enterprise Edition (32-bit)

Windows 7 Professional, Ultimate and Enterprise Edition (64-bit)

Solaris SPARC 10 (64-bit)

AIX 6.1 Power (64-bit)

AIX 7.1 Power (64-bit)

Red Hat Enterprise Linux 5 for x86 (32-bit)

Red Hat Enterprise Linux 6 for x86 (64-bit)

i5/OS V6R1 (OS/400)

i5/OS V7R1 (OS/400)

Recommended Configurations

Software AG provides support for operating system versions, Java versions, browser versions and application server versions supported by their respective vendors. Generally, when a vendor stops supporting an OS version, Java version, browser version or application server version, Software AG will stop supporting that version as of the next JIS service pack level delivered by Software AG. Although it may be technically possible to run a new version of JIS using an unsupported version, Software AG cannot continue to support configurations that are no longer supported by their vendor.

ACE

The ACE interactive development kit has been tested on the following operating systems:

- Windows XP Professional SP3.
- Windows 7.

When developing on Windows 7, in order to install a JIS runtime installation on Unix/OS400, you need to add the %windir%\System32\ftp.exe application to the Windows firewall allowed application list.

Clients

The Java Client has been tested on the following operating systems, browser and Java versions:

- Windows XP Professional SP3.
- Windows 7.
- RedHat Linux 6.

Browser	JRE
IE 7, 8, 9	Oracle JRE 1.6.0 and 1.7.0
Firefox	Oracle JRE 1.6.0 and 1.7.0
Chrome	Oracle JRE 1.6.0 and 1.7.0

When working with a native 64 bit Windows operating system version such as Windows 7, 64 bit, in order to run the JIS Java client, the Java runtime environment must be installed twice, Java 32 bit for Internet Explorer 32 bit and Java 64 bit for Internet Explorer 64 bit.

The XHTML client has been tested with the following operating system and browser versions:

Operating System	Browser
Windows 7/Windows XP	IE 7, IE 8, IE 9
Windows 7/Windows XP	Firefox
Windows 7/Windows XP	Safari
Windows 7/Windows XP	Chrome
RedHat Linux 6	Firefox
Mac OS	Safari

JIS Standalone Server

The JIS standalone server has been tested in the following environments:

Operating System	Java version
Windows 2003	Oracle 1.6.0 32 bit
Windows 2008	Oracle 1.6.0 32 bit
Solaris 10	Oracle 1.6.0 32 bit
AIX 6.1 and 7.1	IBM 1.6.0 32 bit
i5/OS V6R1 and V7R1	IBM 1.6.0 32 bit
RedHat Linux AS5 and AS6	Oracle 1.6.0 32 bit

Note: In order to run JIS on RedHat Linux 64 bit, install the shared object libstdc++-libc6.2-2.so.3 from within the compat-libstdc++-296-2.96-144.el6.i686.rpm package from the RedHat Linux installation media using the command:

```
rpm -i compat-libstdc++-296-2.96-144.el6.i686.rpm
```

J2EE Deployment

The runtime of the JIS XHTML client has been tested for deployment in the following environments:

Application Server or Web Container	**Java Runtime Environment**	**Operating system**
WebSphere 6.1.0.17	IBM JDK 1.5	Windows 2003 Enterprise Edition
WebSphere 6.1.0.17	IBM JDK 1.5	Solaris 10
WebSphere 6.1.0.27	IBM JDK 1.5	RedHat Linux AS5
WebSphere 7.0.0.11	IBM JDK 1.6	Windows 2003 Enterprise Edition

OS400 components

The Innovator components and the DDS compiler have been tested on the following operating systems:

- OS400 V6R1
- OS400 V7R1

Retirement of the Innovator and Studio Components

Following the announcement made in the JIS 9.1.0 Release Notes, all subsequent JIS releases and service packs will no longer provide support for the Innovator and Studio Components.

New Features in Version 9.1.2

Creating screen images from Natural Maps

One of the major strengths of JIS is its ability to create screen images directly from host screen maps. This provides many advantages over creating the screen images from screen captures. Starting from release 9.1.2, JIS now supports creating screen images directly from Software AG Natural map files by integrating the Software AG Natural parser component into the JIS codebase. Natural map files are first converted into JIS SDF standard maps and then to JIS screen images.

JIS supports creating screen images from the following Natural map formats:

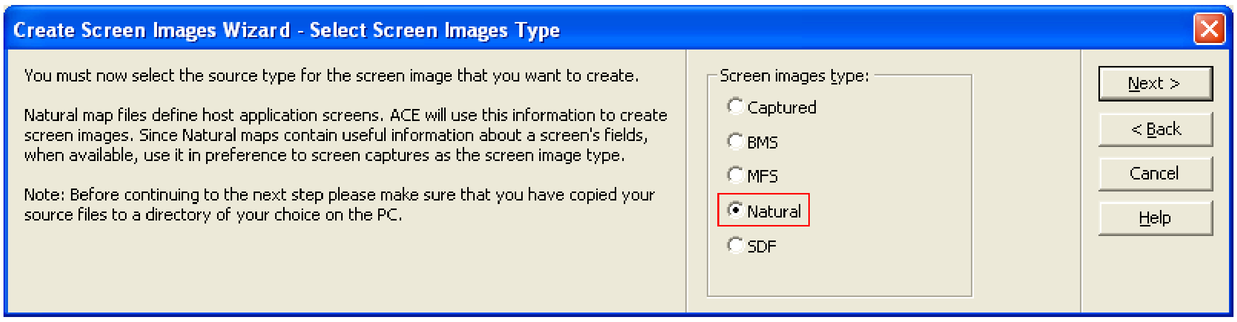
- NSM format – this is the map file source itself which can be imported from the mainframe or from a NaturalOne project.

- NCD format – this map format is generated using the Natural SYSOBJH utility.

As Natural map files do not contain information required for creating function keys and popup window borders, JIS provides additional mechanisms for adding this information to the generated screen images.

Importing Natural maps:

The process of creating screen images from Natural maps is very similar to the process of creating screen images from other Mainframe map formats such as BMS or MFS.


Create Screen Images Wizard - Select Screen Images Type

You must now select the source type for the screen image that you want to create.

Natural map files define host application screens. ACE will use this information to create screen images. Since Natural maps contain useful information about a screen's fields, when available, use it in preference to screen captures as the screen image type.

Note: Before continuing to the next step please make sure that you have copied your source files to a directory of your choice on the PC.

Screen images type: Captured, BMS, MFS, [✓ Natural], SDF

1. In the Create Screen Images Wizard, in the Select Screen Images Type step, select Natural.

2. In the Select Source files screen, select Natural map source files to compile:

   - NCD: where each file represents one or more maps

   - NSM: where each file represents a single map

Creating Popup Windows from Natural Maps

The Natural map does not contain information as to whether the map should be displayed as a popup window in runtime, and what the popup window's borders should be. Therefore, in order to support creating screen images for host popup windows the border of the window must be defined.

Popup windows in Natural are defined using the DEFINE WINDOW command: http://documentation.softwareag.com/natural/nat821mf/sm/definewi.htm. JIS uses properties similar to the ones used by the Natural DEFINE WINDOW command to display the pop-up window border in design time as close as possible as to how it would be displayed by Natural during runtime.

1. In order to achieve this, the following properties need to be specified per window map in the natural_parser.properties file in the <AceRoot> folder. Define the following properties for each window map. The existing properties file provided with the product, provides an example of the required properties:

Property name	**Description**	**Default value**
<Map name>.IS.WINDOW	"TRUE" specifies that the current map represents a window	FALSE
<Map name>.WINDOW.BASE	This property is equivalent to the DEFINE WINDOW command BASE clause. Only the BASE operand3/operand4 format is currently supported. The BASE TOP/BOTTOM LEFT/RIGHT and BASE CURSOR options are not supported.	1/1
<Map name>.WINDOW.SIZE	This property is equivalent to the DEFINE WINDOW command SIZE clause. The options SIZE operand1 * operand2 and SIZE AUTO are supported. The SIZE QUARTER option is not supported.	AUTO
<Map name>.WINDOW.FRAME	This setting is always 3 characters long. The 1st character represents the corner character. The 2nd character represents the horizontal border and the 3rd character represents the vertical border.	Blank border
<Map name>.WINDOW.TITLE	This property is equivalent to the DEFINE WINDOW command TITLE clause.	No title
<Map name>.WINDOW.COLOR	This property is equivalent to the DEFINE WINDOW command FRAMED	The neutral color: "NE"

	(CD=frame-color) clause. The list of possible color values is specified in: http://documentation.softwareag.com/natural/nat821mf/parms/sp_cd.htm	

2. In the Create Screen Images Wizard, select the relevant Natural map file representing the Natural popup window content.

When creating the new subapplication using the New Subapplication wizard, the subapplication will be marked as "host popup" and the resulting host screen will include a popup border based on the window properties specified above.

Handling Function (F) Keys

By default the Natural function key lines appear in the following form on the mainframe screen:

```
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      Help        Exit  Last        Flip                                Canc
```

However, many variations exist including function keys PF13 to PF24 or different function key layouts such as F3=EXIT. The various terminal commands which control the layout of the Natural function key lines are documented here:

http://documentation.softwareag.com/natural/nat821mf/tcom/pcy.htm#PERCENT_YN

In addition, Natural maps do not provide information about the position and layout of the Natural function key lines in runtime.

As JIS relies on the function key information in the screen image in order to identify the screen in runtime and in order to define Buttons, Menu items and Accelerator representations, it provides two different options for displaying the function key lines in the generated screen image:

1. STATIC – the default Natural function key line shown above is displayed on the screen image without the function key's description. The user needs to capture and combine an actual host screen in order to append the function key description to the screen image. Use this mode only if your application always uses the default Natural function keys line. This mode is compatible with the screen images generated by the old mainframe based Natural parser.

2. DYNAMIC – the screen images contain prototype information and the actual function keys are created in runtime – this mode is more flexible and supports most function key layouts. Use this mode when creating a new application.

The type of function key lines displayed in the screen image is controlled by the following specific.ini setting:

```
[NaturalParser]
PFType=STATIC or PFType=DYNAMIC
```

The default value is STATIC however new applications are created with the value preset to DYNAMIC

Displaying function keys using the STATIC option

When using the STATIC option, the natural parser displays the following line exactly 2 rows from the bottom of the screen (line 22 in model 2 screens)

```
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
```

In runtime, JIS expects this line to appear as is, or else the screen won't be identified.

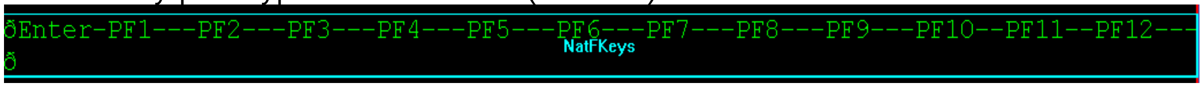
Use the Static function key layout when using Natural maps for existing projects which relied on the old mainframe based JIS Natural parser. In this scenario every map based screen image needs to be combined with a corresponding screen capture in order to overlay the function key descriptions one line from the bottom of the screen (line 23 in model 2 screens).

In order for the static keys pattern to be analyzed correctly all of the steps below must be performed:

1. Set PFType=STATIC as explained above before importing the Natural map.

2. Import the Natural map and compile it into a screen image using the "Create Screen Images" or "Maintain Screen Images" wizards.

3. When creating the new subapplication from the Natural screen image, in the "Select Screen Layout" step choose the "WebLookNatFKeys" screen layout or in the subapplication itself, open "Layout View" and drag the section "NatFKeys" around the function key prototypes in lines 22-23 (model 2).



```
õEnter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
õ
```

4. Capture the screen representing the current Natural map in runtime and combine it with the subapplication created from the map in order to display the function keys together with their description.

   Existing applications, using knowledge base definitions, developed in previous versions of JIS and which did not develop their own pattern definitions for matching Natural Fkeys, should append the knowledge base definitions from the NATURAL.GKB file in <JISRoot>\KB_3270 into their knowledge base list in the specific.ini (note: this is an advanced operation which requires knowledge base administration skills)

Displaying function keys using the DYNAMIC option

When using the DYNAMIC option, the Natural parser generates prototype information for the JIS Dynamic FKeys feature. During design time, JIS uses the prototypes to identify the possible FKey patterns which may appear in runtime. Therefore, there is no need to capture and combine screens.

The supported emulator commands that can be used are:

%YN, %YS, %YP – for the different layouts of the function keys line

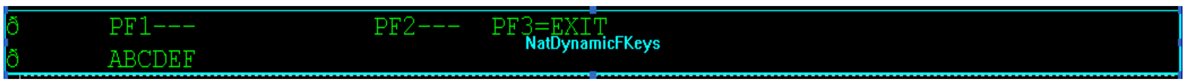
%YA - to display both function key lines

%YF, %YL, %YX – to switch between F1-F12 and F13-F24 function key lines

The Dynamic option is the default option for new applications.

In order for the dynamic keys pattern to be analyzed correctly all of the steps below have to be performed:

1. Set PFType=DYNAMIC as explained above before importing the Natural map.
2. Import the Natural map and compile it into a screen image using the “Create Screen Images” or “Maintain Screen Images” wizards.
3. When creating the new subapplication from the Natural screen image, in the “Select Screen Layout” step choose the “WebLookNatDynamicFKeys” screen layout or in the subapplication itself, open “Layout View” and drag the section “NatDynamicFKeys” around the function key prototypes in lines 22-23 (model 2).


```
ð       PF1---              PF2---   PF3=EXIT
ð       ABCDEF
```

4. Existing applications, using knowledge base definitions, developed in previous versions of JIS should append the knowledge base definitions from the NATURAL.GKB file in <JISRoot>\KB_3270 into their knowledge base list in the specific.ini (note: this is an advanced operation which requires knowledge base administration skills)

Additional Parser Configurations

Additional settings can be specified in specific.ini section [NaturalParser]

MessageLinePosition - Indicates on which line of the screen image to position the message line. The possible values are FIRST and LAST (default: LAST).

Model Type – indicates the screen model of the screen image (default: 2)

MapFileEncoding – specifies the encoding used by the Natural map source file. By default, the operating system default encoding is used.

Note: When importing maps from NaturalONE specify MapFileEncoding=UTF8

SkipWriteCommands - Select this in order not to generate maps that are called with the NATURAL WRITE command, possible values are 0 and 1 (default: 1).

Limitations

The function key lines are always displayed in their default location, two lines above the bottom of the screen (lines 22-23 with screen model 2 for example) assuming that Natural uses the terminal command %YB.

The positioning related terminal commands: %YT, %Ynn and %Y are not supported and will cause the screen not to identify in runtime.

Function key lines and message line inside a popup window, are not supported.

The message line is assumed to be in the last line of the screen (Terminal command %MB) or the first line of the screen.

Importing Natural maps from AS/400 applications is not supported.

Natural Maps and JITGUI

The default JITGUI subapplication has been enhanced to automatically recognize the Natural default function keys layouts F1-F12 and F13-F24.

Simplified HTTPS/SSL Configuration

Improving the Keystore Configuration

In order for the JIS server to use Https and SSL when communicating with the client, the server has to have access to a Java keystore in which the private key and the server certificate are stored.

In previous versions the process of creating the keystore was manual using the keytool command line utility. Now, the JettyKeyStore file, the private key and a test certificate are generated automatically the first time the server is started, thus providing the ability to use HTTPS and SSL out of the box with minimal additional configuration.

Upon startup the server checks if one of the following flags is enabled in the jacadasv.ini:

```
[General]
JavaClientSSLEnabled=1
```

or

```
[Http]
SupportHTTPS=1
```

If so, the server looks for a file named JettyKeyStore in its classpath. If this file exists, the server loads it and uses it as its keystore. This allows users to continue to use their existing keystore or create a keystore with unique properties which cannot be created automatically.

When the file does not exist (which is always the case for a new installation), the server generates a new keystore file in the <JISRootDir>\JacadaFiles\classes folder with the name JettyKeyStore and creates an X509 server certificate based on information provided in the jacadasv.ini [KEYSTORE] section.

The [KEYSTORE] section contains a number of settings which provide the information necessary for creating an X509 certificate:

Domain - represents the network DNS name of the server, this should be the address provided by the client browser when connecting to the server using HTTPS or SSL. For example: www.mydomain.com or sagjacada.eur.ad.sag. Specifying this name correctly is important in order to avoid an HTTPS warning message from the client browser. By default JIS sets this value to the network name of the machine on which the server is running.

The OrganizationalUnit, Organization, City, and State settings are the X509 certificate distinguished name fields which contain free text relevant for the customer's site. The default value of each of these settings is "Unknown". It's important to set these settings correctly as they can be used later for generating a certificate signing request as part of the process of obtaining a valid SSL certificate instead of the auto generated test certificate.

The automatically generated JettyKeyStore has the following properties:

Keystore format: JKS

Keystore password: defaults to the value specified by the jacadasv.ini setting:

```
[HTTP]
KeystorePassword=
```

When the setting is not specified, the default password is "JettyKeyStore".

The automatically generated private key has the following properties:

Alias: server.key

Key algorithm: RSA

Key password: same as the keystore password.

These settings are non configurable.

The keystore is generated using the KeyTool utility provided by the vendor of the Java VM, currently Oracle (SUN) and IBM Java VMs are supported.

Setting up HTTPS Communication between the XHTML Client and the Server

For XHTML users, HTTPS communication is enabled once the JettyKeyStore has been created and the SupportHTTPS=1 setting is defined in the [Http] section of jacadasv.ini. HTTPS communication works by default, by accessing the server on port 8443 and using the following URL:

```
https://<Server Address>:8443/<AppName>-xhtml.html
```

Note that the browser will show a browser specific warning related to the website security certificate. For example: Internet Explorer 7 will show a page titled “There is a problem with this website's security certificate.”. Ignore these warnings and continue to the web site in order to establish an HTTPS connection. For explanation of how to eliminate the warning, see the "Browser Certificate Warning when Connecting to the Server" section below.

Setting up HTTPS Communication between the Java Client and the Server

Java client users, using the Applet parameter <PARAM name = "UseHttp" value = "true">, HTTPS is enabled once the JettyKeyStore has been created and the SupportHTTPS=1 settings is defined. Https communication works by default by loading the launcher Html page from the server on port 8443 using the following URL:

```
https://<Server Address>:8443/<AppName>-signed.html
```

Setting up SSL Connection between the Java Client and the Server

Java client users, using ports communication, which is the default communication method, can now setup SSL communication without writing Java extensions.

When starting the Server, for each server process two ports will be created for SSL communication, in addition to the two existing ports used for plain text communication.

Furthermore, plain text communication can be disabled, thus forcing the user to use the SSL option.

The following jacadasv.ini settings control the SSL configuration:

JavaClientSSLEnabled - enables the SSL communication with the Java client. Possible values: 1, 0 (default value: 1).

JavaClientSSLOnly - when set, disables plain text communication, ensuring that the Java client uses SSL communication. Possible values: 1, 0 (default value: 0).

SSLServerPortRange – determines the ports used for SSL communication. The default range of values that can be used for a single process configuration is 1200-1201. The port range needs to be large enough to allow each server process to allocate two SSL ports just like the allocation process for the plain text ServerPortRange.

By default, both SSL and plain text ports are open on the server side. To configure the Applet to use SSL communication, do one of the following:

Set the Applet parameter:

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

Possible values: true, false (default value: false).

Other clients not using this Applet parameter can still communicate using plain text.

Alternatively, set the JavaClientSSLOnly=1 setting on the server side. This will force the Java client to use SSL.

Note: the following configurations will prevent the client from communicating with the server:

```
JavaClientSSLEnabled=0 and <PARAM name = "UseSSL" value = "true">
```

or

```
JavaClientSSLOnly=1 and <PARAM name = "UseSSL" value = "false">
```

Note: The communication method (Ports or HTTP/s) used by the Java client now depends only on the value of the UseHttp Applet parameter. The UsePorts Applet parameter has been deprecated. Therefore, when <PARAM name = "UseHttp" value = "true"> is set, the client will use HTTP/S communication. Otherwise it will use the default port communication which can now be encrypted using SSL.

Note:

The combination of the settings:

<PARAM name = "UseSSL" value = "true"> and <PARAM name = "UseHttp" value = "true"> is possible but makes no sense in most configurations. It will cause the communication between the client and the proxy servlet to use HTTP or HTTPS and communication between the proxy servlet and the server to use SSL.

Browser Certificate Warning when Connecting to the Server

The JettyKeyStore generated automatically by the server contains an auto generated certificate which is not trusted by any official certificate authority. Therefore when connecting to the server using HTTPS, the browser will issue a warning message. There are two alternatives for eliminating the warning:

1. Manually import the server certificate into the browser. This is a browser specific procedure which tells the browser to trust the server certificate. Each browser uses its own methods for importing the certificate.

2. Generate a certificate signing request and have it signed by a certificate authority recognized by the browser and Java versions. Since JIS relies on standard Java security architecture this should be a standard process which we do not cover in this document.

SSL connection between the server and the host

It is no longer necessary to import the host certificate into the Java Keystore in order to initiate an SSL connection to a secured port defined on the host.

The following ini setting should be used in order for JIS to initiate a secure connection:

```
[GUISys TN3270] or [GUISys TN5250]
SecureHostConnection=1
```

This new setting replaces the old setting, which is currently still supported for backward compatibility.

```
SocketImplFactory=cst.server.comm.CSTSSLSocketFactory
```

IPv6 Support

JIS now supports using Internet Protocol version 6 for all runtime components including the client browser, standalone server and mainframe. All IP addresses can now be specified using the IPv6 address format.

Note: When using the XHTML RedirectionProxy and specifying server address using IPv6 address format the address must be surrounded with square brackets.

Example:

```
<Settings>
  <JacadaServerAddress>
    <IPAddress>[fe80::21c:23ff:fe31:8268]</IPAddress>
  </JacadaServerAddress>
...
<Settings>
```

Limitations:

Capturing screens from ACE is not supported when the host address uses IPv6 address format.

Specifying a Folder where the Java Client Log File will be Saved

It is now possible to determine that you want to save the Client log file in a specific folder. To do this set the following Applet parameter:

```
<PARAM name ="DebugFileFolder" value="<path to a local file system folder">
```

For example:

```
<PARAM name ="DebugFileFolder" value="c:\temp">
```

If the specified folder does not exist on the local workstation, the log file will be created in the operating system temp folder.

The Java console displays the following message indicating the location of the log file:

Client log file name is: c:\temp\debug_1317653980569.log

In order to use this feature you must use the signed Java client Applet.

Logging Messages Improvements

The following messages have been added to the logger:

- When receiving the version mismatch page for the Java client, the client log will now include the time stamps of the client code and server code, thus providing better understanding of the problem.

- All uncaught exceptions are now logged in the client log when using the signed Java client Applet and in the server logs.

- When running the XHTML pages inside the browser, JavaScript exceptions are now dispatched to the server and correctly logged to the server log by default.

- The server log now identifies the Linux operating system.

- The product now represents debug filters internally using a java.lang.Enum instead of the old implementation which relied on String constants. Therefore, when using "Method Debugging" it is required to generate the runtime again and when using code extensions which utilize debug filters, this code will need to be re-compiled. This operation is performed once, after updating the version.

Proxy Servlet Improvements

When accessing the servlet monitoring page using the URL /JISProxyServlet, the list of active connections is now printed to the server log. This can be useful in order to compare the number of open connections displayed by the proxy servlet with the number of open sessions displayed by the JIS Administrator. Open the server log and search for "List of open server connections".

Updated JIS Perl to Version 5.12.2.0

The Perl distribution used by JIS has been updated to Strawberry Perl 5.12.2.0.

Session Dump Improvements

The following improvements were made in the session dump mechanism:

1. On the Java client the dump is now printed to both the Java console and the log file.

2. On the server the dump can be turned off completely using the following runtime ini setting:

```
[SessionCoreDump]

IsEnabled=0
```

3. Additional exceptions are now recorded in the dump.

Access Log

The NCSA access log contains a record of all inbound client requests that the embedded Jetty web server handles. All of the messages written to the access log are in NCSA format which is a standard format used by web servers and supported by common log analyzing tools.

The access log complements the product server log and makes it simpler to identify problems such as:

1. Response errors.

2. Slow response times.

3. Sessions jumping between servers.

4. Cookie related problems.

Example:

```
localhost 0:0:0:0:0:0:0:1 - - [13/Nov/2011:15:28:15 +0200] "GET /XHTMLV9-xhtml.html
HTTP/1.1" 304 0 "-" "Mozilla/5.0 (Windows NT 5.1) AppleWebKit/535.2 (KHTML, like
Gecko) Chrome/15.0.874.106 Safari/535.2" - 0

localhost 0:0:0:0:0:0:0:1 - - [13/Nov/2011:15:28:15 +0200] "GET
/Xhtml?JacadaApplicationName=XHTMLV9&Language=fr HTTP/1.1" 200 9142
"http://localhost:28080/XHTMLV9-xhtml.html" "Mozilla/5.0 (Windows NT 5.1)
AppleWebKit/535.2 (KHTML, like Gecko) Chrome/15.0.874.106 Safari/535.2" - 12063

localhost 0:0:0:0:0:0:0:1 - - [13/Nov/2011:15:28:28 +0200] "GET
/XhtmlCSS?JacadaApplicationName=XHTMLV9&SessionId=1638907054&LibraryName=XHTMLV9&Suba
pplName=_CSS_LOGIN&CrcCode=1288698078&JBS=8126b2c9836be51cae537e50f369fa00299afc79777
274d0 HTTP/1.1" 200 4282
"http://localhost:28080/Xhtml?JacadaApplicationName=XHTMLV9&Language=fr" "Mozilla/5.0
(Windows NT 5.1) AppleWebKit/535.2 (KHTML, like Gecko) Chrome/15.0.874.106
Safari/535.2" - 15
```

The lines above shows a typical sequence of requests generated when starting a new XHTML session.

The parameters logged are:

- Server name
- Client address
- Username - currently not supported
- Date and Time
- Http method
- URI
- Protocol version
- Response status
- Response length [bytes]
- Referrer
- User-agent
- Response time [ms]

To activate access logging, either run the server with debug level 70 or higher or add the new "ACCESS" filter to the list of debug filters.

There is one access log per server process, the log is created in the same folder as the server log. The log name is of the following format:

access<process_alias>_yyyy_mm_dd.log

For example the access log for server process 1.1 for the date November 13th, 2011 is named:

access_1.1_2011_11_13.log

The access log is rolled over every 24 hours and is kept for 14 days. There is no limit on the size of the access log. When restarting the server, the new access log is appended to the existing access log.

Pattern Matching according to Character Attributes

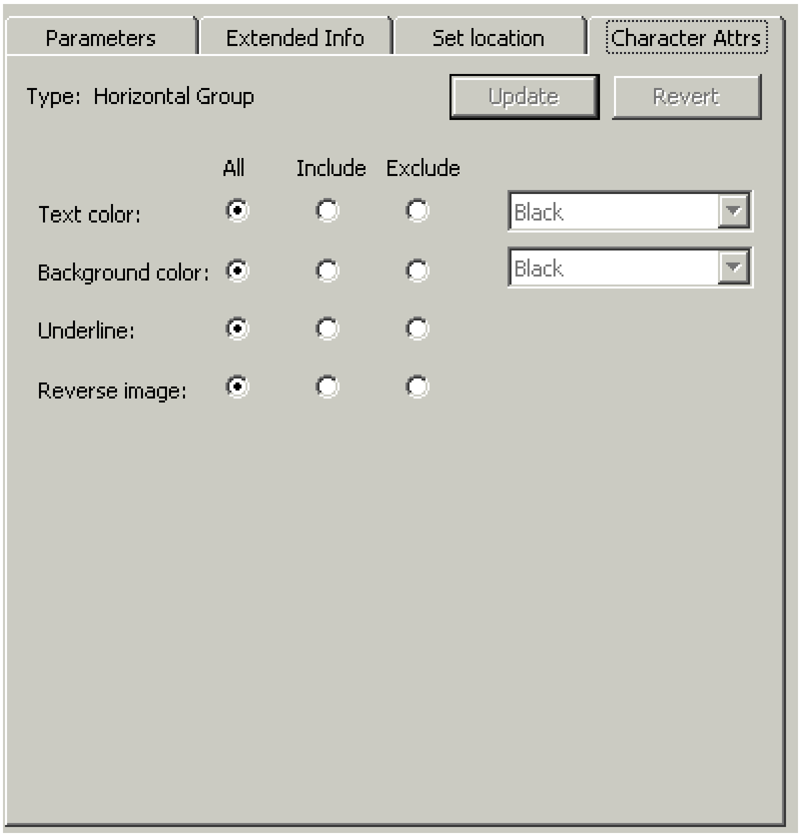
ACE always supported pattern matching using the following character attributes:

- Text color: foreground color of the text on the host screen.
- Background color: background color of the text on the host screen.

- Underline
- Reverse image

However, the knowledge base user interface did not support defining pattern definitions based on these attributes. Now the user interface has been added for these character attributes. The new user interface is only enabled for the “Horizontal group” and “Vertical group” pattern definition types.

A new tab was added to the "Pattern Definitions View" dialog box.



Type: Horizontal Group

	All	Include	Exclude	
Text color:	●	○	○	Black
Background color:	●	○	○	Black
Underline:	●	○	○	
Reverse image:	●	○	○	

Each line in the dialog represents an attribute that could be set for this definition:

"All" - the definition will have no effect on pattern matching (this is the default value).

"Include" - the pattern definition will match only if the matched location on the screen includes the attribute definitions.

"Exclude" - the pattern definition will match only if the matched location on the screen does not include the attribute definitions.

The color combo boxes are disabled when the corresponding "All" radio buttons are checked.

Limitations

When matching patterns according to the "Reverse image" attribute, define "Include" "Reverse image" and "Exclude" "Underline".

Detailed Description of Version 9.1.2 Fixes

Note: The number at the beginning of each ticket item, represents the external support system incident or internal tracking number.

Installation

SI-1033305: When updating the operating system path, the installation now uses the time bound SendMessageTimeout Windows API instead of the SendMessage API to prevent the installation from freezing when the path cannot be updated.

JAVA Client

JIS-647: When running the Java client as an application, the parameters equivalent to the Applet parameters in the launcher Html are read from the file params.txt located in the folder <JISRoot>\JacadaFiles\classes\appls\<AppName>\user. As the Applet Parameters were case insensitive and the parameters in params.txt were case sensitive, copying parameters between the two configurations led to confusing results. Now the parameters in params.txt are also case insensitive.

JIS-640: The Java client JavaDoc is now installed only when Java client is available in the CD Key

SI-5042482: The font resource loading has been fixed.

SI-5055856: Sending a client reply message starting with the value 0x0C caused the session to disconnect.

JIS-628: The default value of the UseNewHTML Applet parameter is now set to "false".

SI-5041270: The Java internal typeahead mechanism is now disabled.

SI-5059124: Since we introduced the ability to log information to the file system, by default all log messages are written to both the Java console and the client log file in the temp folder. However, some exception stack traces and the client session dump were only written to the file and not to the console. This has been fixed, and now all exceptions and the client session dump are logged both to the Java console and to the client log.

SI-5061479: When adding a custom copyright message to your application in ace.ini or ace400.ini which includes the © ASCII 0xA9 character, the © symbol was not displayed correctly in the Java client Help About dialog also causing problems localizing the copyright message.

SI-5056228: This issue is regarding the code sample in Java_Client.pdf page 264 "Methods for Controlling the Java Client Application".

Make sure that any updates to the Java client window are performed using the AWT event dispatch thread, so that the code sample on page 268 inside the run() method should look like this:

```
...
EventQueue.invokeAndWait(
      new Runnable() {
         public void run() {
            login.userField.setText("guest");
            login.passwordField.setText("foobar");
         }
   });
...
```

SI-5058523: All HTTP communication is now run within privileged action context.

XHTML Client

SI-5057726: In a Y/N checkbox, when the value on the host was "Y" the checkbox was still displayed as unchecked.

JIS-599: When using OptimizeStyleAttributes=1, folded tables were not displayed correctly.

JIS-1043: Line and border colors in Safari were incorrect.

JIS-1038: The page size optimization feature did not work correctly with the redirection proxy.

SI-1033559: Submitting a page while the focus was on a Combobox or on a label within a table, did not always send the correct focused control to the server.

JIS-1025: There was a problem when deploying an application to WebLogic and using page size optimization (OptimizeStyleAttributes=1).

JIS-610: In previous versions the XHTML RedirectionProxy contained Sun specific code which prevented it from using the IBM JVM. This has been fixed.

JIS-676: Deployment of J2EE application on weblogic 10.3.4 failed.

SI-5015800: It is now possible to include the underscore character in the name of a table component.

SI-1044586: An exception related to using the PrintString method after calling the Close method on an external output stream has been fixed.

JIS-1102: When using the Chrome browser redundant "Keepalive" messages were sent by the browser.

SI-1034309: In a window which included a tab component, the tabbing order did not work correctly. As a result of this fix, the tab folder titles are no longer part of the tabbing order.

Server

SI-5037488: The return value of the default UserRefreshSubApplication of the NO_ATTRS screen has been changed to False, since returning True prevents the screen from refreshing, causing various problems.

SI-5036973: When using the MaintainFormatTableEntryOn5250FieldSplit ini setting, some fields were incorrectly displayed on the emulator screen.

JIS-629: In the jacadasv.ini file, the ProcessRespawnEnabled setting and the settings in [ProcessCheck] section are no longer supported as these caused stability and security risks.

SI-5061285: There was a memory leak related to JMX when running the server using Java 1.5. JMX is now only enabled when using Java 1.6 and above.

JIS-1075: In the JIS Administrator, the parameter transaction per minute sometimes displayed zero even though the server was actively executing transactions.

Innovator

SI-5015378: The table selection was not removed even when the table was not in focus.

Limitations

Limitations for JIS version 9.1.2

- When creating a screen image from any SDF, during design time, the field colors used by the color table are always considered to be Green regardless of the real field color. If you use a screen capture the color table works correctly. In runtime, the color table is correct.

- JMX is not supported when running the server using Java version 1.5.

- XHTML host printing: You need to click twice on the Connect/Disconnect button in order to connect/disconnect the printer from the host.

- When running the Java client un-signed Applet and the JISAdminServlet, an Exception related to the crossdomain.xml file is logged to the Java console.

- When running the JISAdminServlet the online help dialogs are no longer available.

- When clicking on the 'X' button to close the server console window, though the window is closed, not all server processes are terminated. We recommend that you always close the server using the QUIT command or using the JIS administrator.

- When using IE8 or higher to run a JIS XHTML application which is deployed to an application server, it is not possible to open more than one JIS session from the same browser window.

- It is not possible to run the JIS server using a 64 bit JRE. Use 32 bit JRE instead.

- The JIS common installation for J2EE deployment cannot be installed on a Windows 2008 64 bit machine.

- When running the JIS server as a Windows service, when stopping the JIS server from the administrator utility, the service is still displayed as 'started' in the Windows services panel. Stop the service from the services panel to clear out this inconsistency.

New Features in Version 9.1.1

XHTML

The XHTML client is now supported on Mac OS and Linux operating systems and Safari and Chrome browsers. See the “Recommended Configurations” section for specific details as to which configurations are recommended and see the “Limitations” section regarding limitations when using these configurations.

Note: In order to use the Mainframe function keys (F1-F24), when using the Safari browser on Mac OS, open the “System Preferences” dialog box, select “Keyboard” and verify that the “Use all F1,F2, etc…” checkbox is checked.

Server Changes

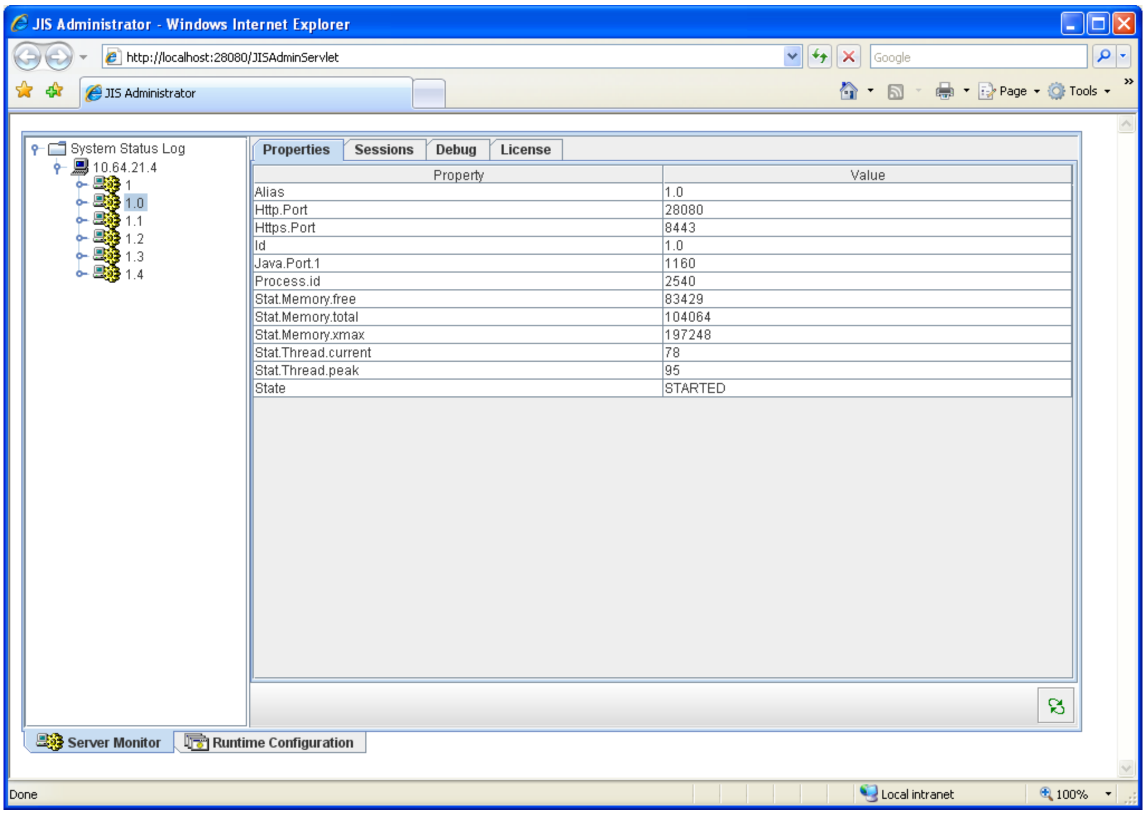
The Server start up, shutdown and restart times have been shortened. In order to achieve this, the following changes were made:

- The Node Registry component no longer runs as a separate Java process. Instead the Node Registry now runs within one of the other server processes. This change improves startup time and removes the need to specify the path to the jacadasv.policy file using the -Djava.security.policy flag.
- Server quit time was reduced by approximately three seconds.
- The Administrator tool now displays the Integrator (1.0) process immediately after it is fully started. Previously the Integrator process only showed up in the monitoring tool, 60 seconds after it was fully started.

Monitoring Improvements

A JIS server deployment is comprised of one or more JIS work processes. Each process represents a Java virtual machine operating system process. The JIS administrator utility is now able to provide environment and performance indicators related to the underlying Java virtual machine. Use these indicators to monitor the status of the underlying Java virtual machine.

New process attributes:



System Status Log
- 10.64.21.4
  - 1
  - 1.0
  - 1.1
  - 1.2
  - 1.3
  - 1.4

Property	Value
Alias	1.0
Http.Port	28080
Https.Port	8443
Id	1.0
Java.Port.1	1160
Process.id	2540
Stat.Memory.free	83429
Stat.Memory.total	104064
Stat.Memory.xmax	197248
Stat.Thread.current	78
Stat.Thread.peak	95
State	STARTED

Process.id: Specifies the operating system process ID - useful for identifying the specific Java virtual machine process in the Windows task manager or using the Unix ps command.

Stat.Memory.free - the amount of free memory, out of the current heap memory size (specified in Kilobytes).

Stat.Memory.total - the current heap memory size (specified in Kilobytes).

Stat.Memory.xmax - the maximum allowed heap memory size, as defined by the Java -mx command line parameter (specified in Kilobytes).

Stat.Thread.current - the current number of operating system threads used by the Java virtual machine.

Stat.Thread.peak - the number of operating system threads used by the Java virtual machine at peak usage since the server started.

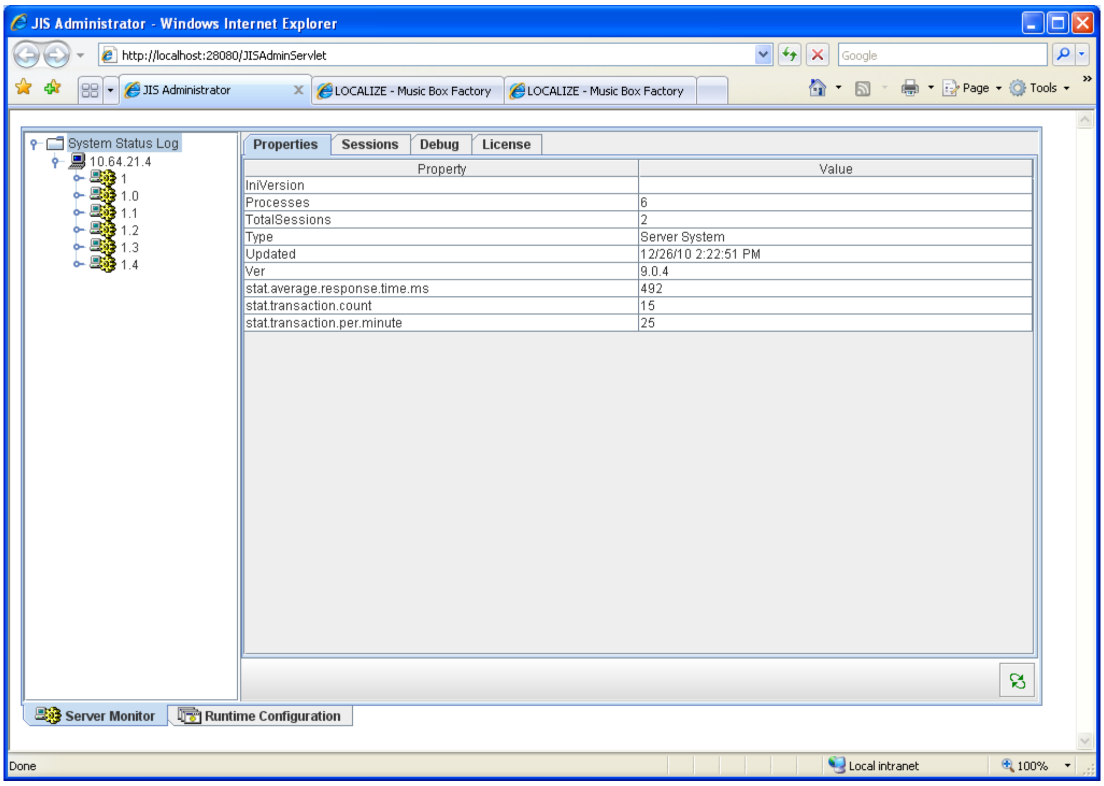
Best practices:

If for a given server process, Stat.Memory.total equals Stat.Memory.xmax and Stat.Memory.free is less than 10% of Stat.Memory.total, then the server process is at risk of running out of heap memory. To mitigate this risk, increase the memory heap size of the specific process using the –mx flag or allocate more work processes on the machine by increasing the MaxProcesses setting.

Before allocating more memory to a process, always make sure the server machine itself is not running out of memory.

A single Java process has limited capacity for running operating system threads, the larger the heap memory the smaller the number of threads available for the Java process. Use the following rule of thumb: if for a given server process, Stat.Thread.peak increases above 2000, consider allocating more Java processes on the server machine.

New system status attributes:



Property	Value
IniVersion	
Processes	6
TotalSessions	2
Type	Server System
Updated	12/26/10 2:22:51 PM
Ver	9.0.4
stat.average.response.time.ms	492
stat.transaction.count	15
stat.transaction.per.minute	25

The JIS administrator now monitors the total number of transactions performed by the server at any given moment. A transaction is defined as the unit of work starting by a client action or host action and ending when the complete response is written to the client. In most cases a transaction consists of a single Mainframe screen transition.

stat.average.response.times.ms – the average server response time in milliseconds. The response time is measured from the time a client request was received by the server and until the response has been fully written back to the client. Therefore this value includes any think time caused by the host and the communication between the server and the host but does not include any think time caused by the client browser or communication between the client and the server. Typically a value of more than 2000 (2 seconds) indicates a performance tuning problem.

stat.transaction.count – the total number of transactions since the server was started. Use this parameter to evaluate the total load on the server and to make sure work is equally distributed between servers in a multi-server configuration (this parameter is not implemented when deploying the application as an .ear file)

Stat.transction.per.minute – the current number of transactions per minute. Measuring this parameter is especially important during peak hours and during server loadtest. You can compare the value of this parameter with the Software AG benchmark results (this parameter is not implemented when deploying the application as an .ear file)

The new performance indicators are exposed in the following configurations:

Standalone JAM, JAM in J2EE, JISAdminServlet, JMX and Java code acting as a JMX client.

Note that a JMX enabled monitoring tool may monitor these parameters over time and allow you to chart the data and define alerts.

Localization Improvements

The following localization improvements have been made to Java Client localization support:

- A new parameter for specifying the encoding of the localization resource file has been added: "ResourceFileEncoding". This parameter is necessary when the encoding used when creating the resource file is different than the encoding used by the client workstation.

  Examples:

  To read a resource file encoded as UTF-8. This is the recommended encoding:

  <PARAM name = "ResourceFileEncoding" value = "UTF-8">

  To read the resource file using simplified Chinese encoding:

  <PARAM name = "ResourceFileEncoding" value = "gbk">

  To read the resource file using simplified Japanese encoding:

  <PARAM name = "ResourceFileEncoding" value = "sjis">

- Text of dynamic menu items is now translated according to the resource file.
- Text labels in the Help-About dialog box can now be translated according to the resource file.

The following localization improvement has been made to both the Java Client and XHTML localization support:

The original string and the translated string can now include multiple appearances of the equal sign '=' and the quotes sign '"'.

ACE

Creation of the runtime installation is only possible for platforms for which runtime was generated.

Runtime Installation

It is now possible to install the JIS runtime installation on Windows to a path which includes spaces. For example: c:\program files\<company name>\<product name>.

This is currently not supported on Unix and AS/400.

Java Client Improvements

Mixed code warning displayed by all versions of JIS when using Java 1.6.0_19 and higher, is no longer displayed.

The clfull-signed.jar and clbase-signed.jar files are now digitally signed and time stamped; hence their signature will continue to be valid after the certificate used for signing the files has expired.

The following limitations have been removed when running the Java client as an application:

1. Link controls are now operational for activating methods (but not for opening a browser URL).

2. The params.txt file is no longer locked for editing while the application is running.

GUI Printing improvements:

- Images in popup windows are now printed correctly.
- Some deprecated APIs have been replaced and logging messages have been improved.

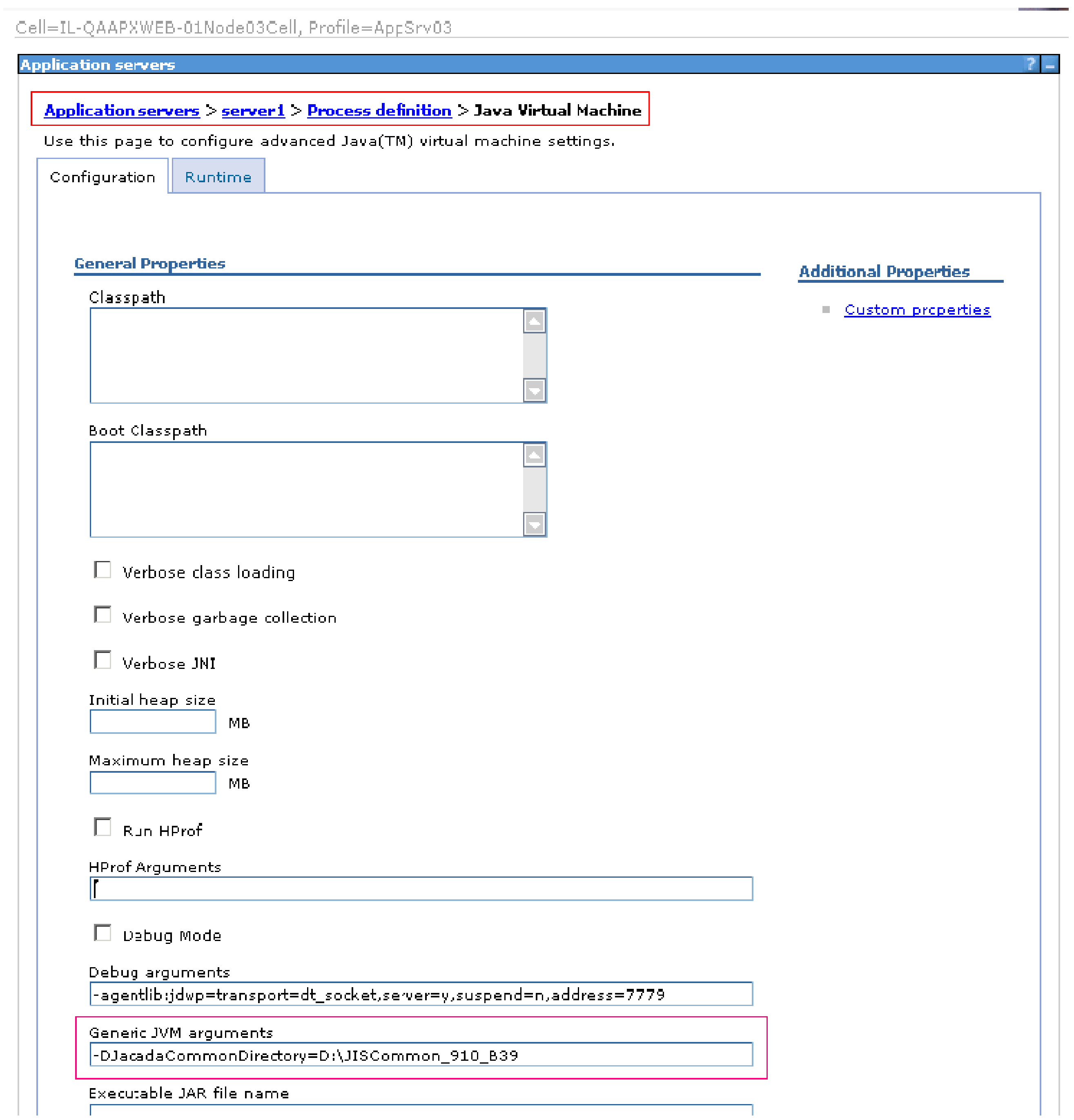
New Features in Version 9.1

WebSphere 7 Support

JIS has been tested using WebSphere 7.0.0.11 on Windows 2003. Deploying a JIS application into WebSphere 7 requires additional configuration:

Copy all the jar files from <JISCommon>\lib to <WAS_HOME>\lib\ext, this operation should be repeated every time the JIS common installation is updated.

Add the following argument to the “Generic JVM arguments” field in the Java Virtual Machine setting panel: -DJacadaCommonDirectory=<Installation directory of JISCommon>, see attached example.


Cell=IL-QAAPXWEB-01Node03Cell, Profile=AppSrv03

Application servers

Application servers > server1 > Process definition > Java Virtual Machine

Use this page to configure advanced Java(TM) virtual machine settings.

General Properties

Classpath

Boot Classpath

Verbose class loading

Verbose garbage collection

Verbose JNI

Initial heap size MB

Maximum heap size MB

Run HProf

HProf Arguments

Debug Mode

Debug arguments
-agentlib:jdwp=transport=dt_socket,server=y,suspend=n,address=7779

Generic JVM arguments
-DJacadaCommonDirectory=D:\JISCommon_910_B39

Executable JAR file name

Additional Properties

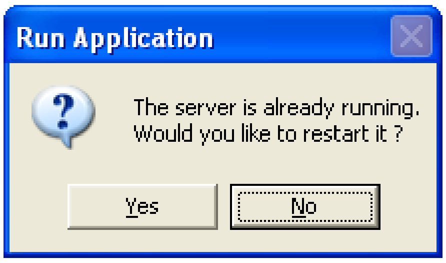
Custom properties

Usability Improvements in the "Generate Runtime" and "Run Application" Wizards

The Generate Runtime functionality has been improved to allow generating the runtime while the server is running, enabling the user to continue using the existing runtime while generating a new version of the runtime.

Run Application Wizard Improvements:

After clicking Finish in the last screen of the Run Application Wizard, JIS asks you whether you would like to restart the server.



Run Application

The server is already running.
Would you like to restart it ?

Restarting the server enables launching the updated application in a new browser window (previously the server was not restarted and the application displayed in the browser did not reflect the changes made).

Additional improvements:

- Default compilation batch size was increased from 30 classes to 90 classes.
- The source and target release of the compiled application classes changed from 1.4 to 1.5. This allows users to write code extensions which rely on Java 5 specific syntax.
- When generating an XHTML client, the obsolete and confusing static HTML files are no longer generated in the <JISRoot>\JacadaFiles\classes\appls\<ApplName>\xhtml\templates\original folder. Users may delete existing old files in this folder to reduce the size of the runtime installation.

Changing Default Settings

In previous releases, after clicking Finish in the last screen of the Run Application Wizard, the default browser associated with the .html extension was opened. This approach which had several drawbacks has been abandoned. Instead, by default, the browser opened, is the browser specified in the following path “C:\Program Files\Internet Explorer\iexplore.exe”. This path can be customized using the ini setting:

```
[RunApplicationWizard]
BrowserCommandLine=<command line for the browser application>
```

Examples:

```
To run the application using Firefox use the following specific.ini setting:
[RunApplicationWizard]
BrowserCommandLine="C:\Program Files\Mozilla Firefox\firefox.exe" -new-window
```

To run the application using Internet Explorer 32 bit on a Windows 64 bit operating system use the following specific.ini setting:

```
BrowserCommandLine="c:\Program Files (x86)\Internet Explorer\iexplore.exe"
```

Maintaining Backward Compatibility

By default now, the server loads native resources, such as .dlr files, using a Java class loader, instead of as platform specific memory mapped files. This prevents the server from locking the resources thus allowing to generate runtime while the server is running. It is possible to change the default behavior in runtime to maintain backward compatibility. Use the following jacadasv.ini setting:

```
[GeneralParameters]
LoadNativeResourcesUsingJava=0
```

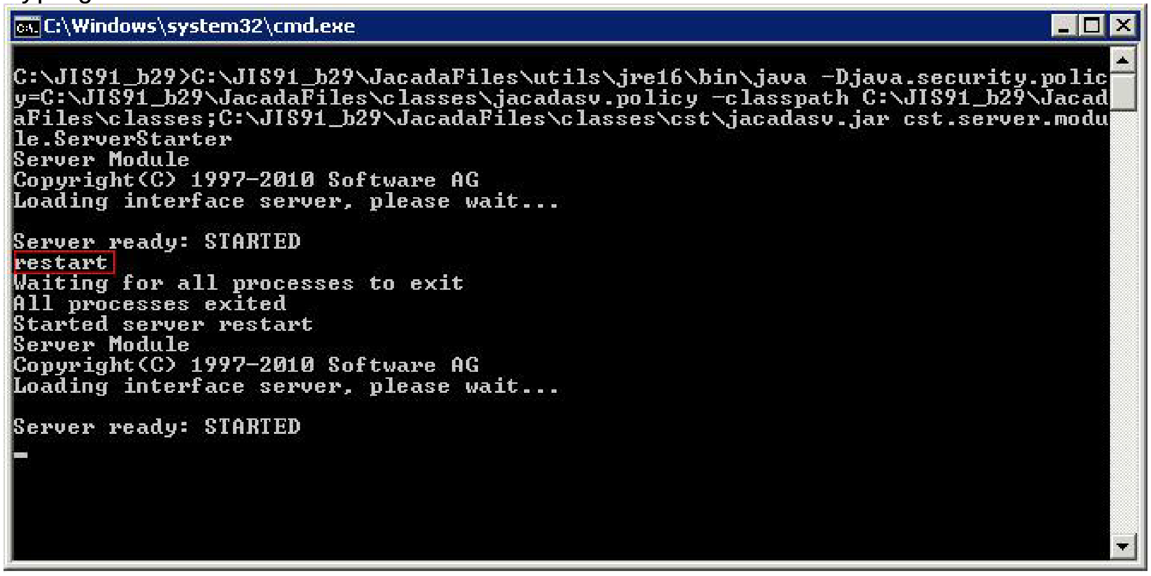
When using this setting you will not be able to generate runtime while the server is running.

Restarting the JIS Server

A new mechanism enables restarting the JIS server. Restarting the server is useful when updating a new version of the application or in order to reload configuration changes which require restarting the server.

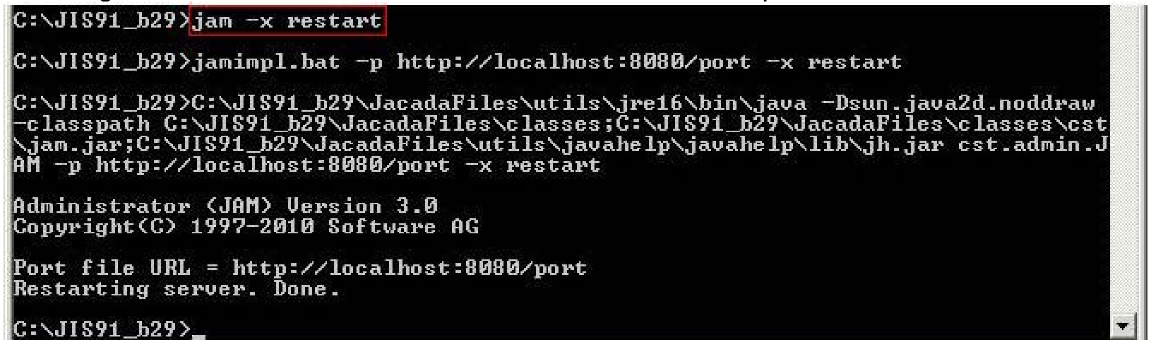
This Restart command can be invoked in the following ways:

Typing RESTART in the server console.



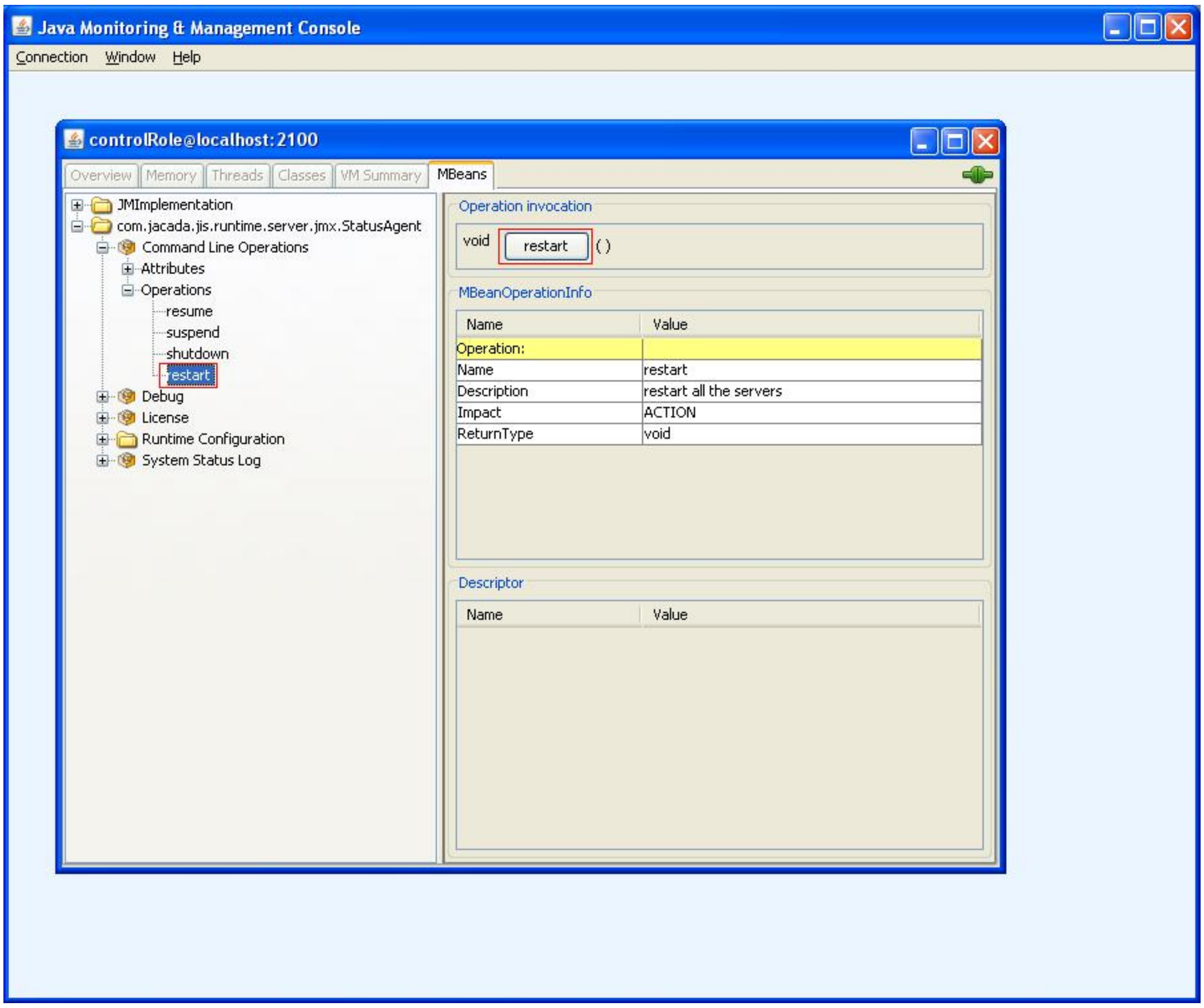
```
C:\JIS91_b29>C:\JIS91_b29\JacadaFiles\utils\jre16\bin\java -Djava.security.polic
y=C:\JIS91_b29\JacadaFiles\classes\jacadasv.policy -classpath C:\JIS91_b29\Jacad
aFiles\classes;C:\JIS91_b29\JacadaFiles\classes\cst\jacadasv.jar cst.server.modu
le.ServerStarter
Server Module
Copyright(C) 1997-2010 Software AG
Loading interface server, please wait...

Server ready: STARTED
restart
Waiting for all processes to exit
All processes exited
Started server restart
Server Module
Copyright(C) 1997-2010 Software AG
Loading interface server, please wait...

Server ready: STARTED
```

Running the restart command via JAM's command line operations.



```
C:\JIS91_b29>jam -x restart

C:\JIS91_b29>jamimpl.bat -p http://localhost:8080/port -x restart

C:\JIS91_b29>C:\JIS91_b29\JacadaFiles\utils\jre16\bin\java -Dsun.java2d.noddraw
-classpath C:\JIS91_b29\JacadaFiles\classes;C:\JIS91_b29\JacadaFiles\classes\cst
\jam.jar;C:\JIS91_b29\JacadaFiles\utils\javahelp\javahelp\lib\jh.jar cst.admin.J
AM -p http://localhost:8080/port -x restart

Administrator (JAM) Version 3.0
Copyright(C) 1997-2010 Software AG

Port file URL = http://localhost:8080/port
Restarting server. Done.

C:\JIS91_b29>
```

Executing the Restart command using a JMX enabled monitoring tool or from a Java class which uses JMX code.



Name	Value
Operation:	
Name	restart
Description	restart all the servers
Impact	ACTION
ReturnType	void

Name	Value

JMX Support

The JMX technology provides the tools for building distributed, Web-based, modular and dynamic solutions for managing and monitoring applications. By design, this standard is suitable for adapting legacy systems, implementing new management and monitoring solutions.

JIS now enables performing administration activities including session monitoring, application configuration and server operations using JMX (previously these activities were available only via the standalone JIS Administrator tool). This allows data management (and data viewing) using monitoring tools which support JMX such as JConsole and/or by writing dedicated Java code.

In order to use JMX enable the XML server in the jacadasv.ini file:

```
[LogClasses]
XMLServer=

[XMLServer]
Enable=1
TimerTick=
```

Note: JMX is supported only when running the standalone JIS Server, and not when using J2EE deployment.

MBeans are *managed beans*, Java objects that represent resources to be managed. Data shown and managed in the standalone JIS Administrator tool is exposed by creating matching JMX's MBeans. All the MBeans exposed by JIS are defined in the object-name root com.jacada.jis.runtime.server.log.StatusAgent and are categorized according to the data and operations they expose.

Following is a detailed list of the configuration data and administrative operations exposed using JMX:

System Status Log: combines read only information for JIS servers, processes, applications and sessions. The information encompasses the same attributes shown in JIS Administrator's properties tables and is sorted in the same hierarchical tree-like topology (Root->Servers->Processes->Applications->Sessions).

Running Sessions: lists information about the currently running sessions displayed as a list, and allows executing operations such as closing sessions and changing the debug level for a specific session.

Debug: contains editable settings that are included in JIS Administrator's Debug panel (Debug level, Log file size, Number of log files and Log directory). The Debug MBean also exposes operations such as placing a message in the log file, clearing the log file and saving the debug settings to the ini file for future use.

License: contains read-only attributes that are shown in JIS Administrator's License panel. Also allows replacing the current license file by specifying the location of a different license file.

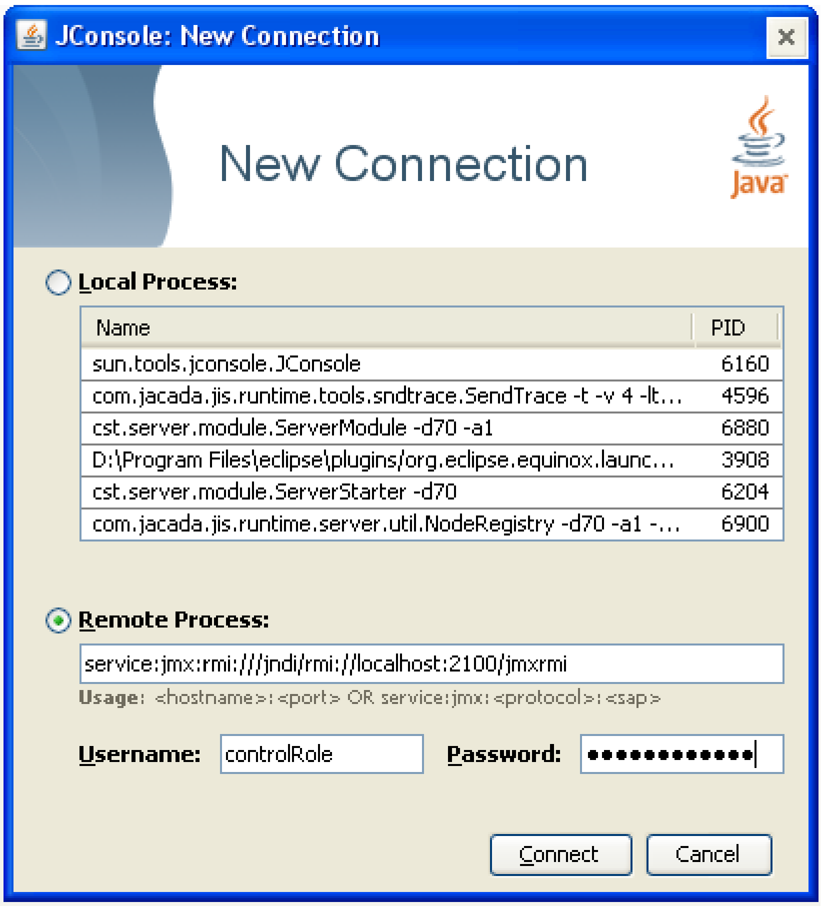
Runtime configuration: allows setting the application's ini file configuration, as done in the JIS Administrator's Runtime Configuration view. The data is sorted in a hierarchical topology - each application deployed on the server includes MBeans per each of the application's configuration sections, each of the sections includes a set of editable parameters.

Command line operations: allows performing the same operations that can be invoked via JIS Administrator command line interface - Shutdown, Restart, Suspend, Resume and Status.

Connecting to the server using a JMX client application

You can connect to the JMX server using/via the client or using Java code.

Connecting via the client:



Local Process:

Name	PID
sun.tools.jconsole.JConsole	6160
com.jacada.jis.runtime.tools.sndtrace.SendTrace -t -v 4 -lt...	4596
cst.server.module.ServerModule -d70 -a1	6880
D:\Program Files\eclipse\plugins/org.eclipse.equinox.launc...	3908
cst.server.module.ServerStarter -d70	6204
com.jacada.jis.runtime.server.util.NodeRegistry -d70 -a1 -...	6900

Remote Process: service:jmx:rmi:///jndi/rmi://localhost:2100/jmxrmi

Usage: <hostname>:<port> OR service:jmx:<protocol>:<sap>

Username: controlRole Password: ••••••••••••

Log remotely to service:jmx:rmi:///jndi/rmi://<hostname>:<rmi port>/jmxrmi, where hostname is the IP address or hostname of the running JIS server, and the port is the port configured in the server registry node (this is the first port specified by the jacadasv.ini [GeneralParameters] RegistryPortRange ini setting).

Enter a username and password. Two users are defined by default: a read only user (username: monitorRole, password: monitorRole) and a user with "write" permissions (username: controlRole, password: controlRole). To change the default usernames and passwords, edit the \classes\jmxremote.password and \classes\jmxremote.access files and make the necessary changes.

Connecting using Java code:

Refer to a number of Java code examples (Appendix A) which demonstrate how to use JMX code to administrate the server.

All examples contain pure Java code and do not rely on any product or 3rd party Jar files.

XHTML Page Size Optimization Improvements

The page size optimization feature was first introduced in JIS 9.0.4 in order to reduce the page size generated by JIS (refer to the JIS 9.0.4 release notes for more information).

The following improvements have been made to the optimization process:

The optimized CSS for sub-applications which contain dynamic controls, such as the JITGUI sub-application, is now generated every time the sub-application is accessed and not only the first time it is accessed.

The optimized CSS is now generated after the server side XHTML extensions finish executing so that it reflects changes made to the page by code extensions.

It is now possible to instruct JIS to generate a new optimized CSS for sub-applications where the page structure has been modified using a code extension. This is done by calling the context.reOptimizeSubApplication() api from the onPageLoad extension:

```
public void onPageLoad(OnPageLoadContext context) {
     ... code changes which affect the style of the specific page instance …
     context.reOptimizeSubApplication();
}
```

Server Log

JIS server log file now uses file renaming when the current log file reaches its maximum size. Once the active log file has reached the maximum size limit, the file is renamed and the revision number is added to the file name. A new log file is created with the original name.

Example:

Start the server allowing each server process to create 6 log files of up to 100MB in size. Use the following command:

jacadasv -b5 -m100000000

-b5 indicates to the server to keep 5 revisions of each process log file in addition to the current process log file.

-m100000000 value in bytes. Defines the maximum size of a single log file to be approximately 100MB.

As a result when running the server over a period of time, the following files are created for the root process:

```
06/27/2010  05:01 PM        22,802,414 debug_1.log
06/27/2010  05:01 PM        99,999,861 debug_1.Rev1.log
06/27/2010  05:01 PM        99,999,966 debug_1.Rev2.log
06/27/2010  05:00 PM        99,999,876 debug_1.Rev3.log
06/27/2010  05:00 PM        99,999,966 debug_1.Rev4.log
06/27/2010  05:00 PM        99,999,966 debug_1.Rev5.log
```

When the size of the debug_1.log file reaches 100MB:

```
debug_1.Rev5.log is deleted
debug_1.Rev4.log is renamed to debug_1.Rev5.log
debug_1.Rev3.log is renamed to debug_1.Rev4.log
debug_1.Rev2.log is renamed to debug_1.Rev3.log
debug_1.Rev1.log is renamed to debug_1.Rev2.log
debug_1.log is renamed to debug_1.Rev1.log
```

The server continues logging into a newly created debug_1.log and so on.

For process 1.3, for example, the log files would be named debug_1.3.log, debug_1.3.rev1.log, …, debug_1.3.rev5.log

Java Client Log

The Java Client log in now written by default to a log file named debug_<timestamp>.log in the %TEMP% folder on the local workstation and not just to the Java console. This can be controlled using the Java Applet parameter DebugFile.

The possible values are:

<PARAM name = "DebugFile" value = "1"> to write log messages to the Java console only, as in previous versions. The drawbacks of this setting are that the log file size is limited and there is an increase in memory consumption.

<PARAM name = "DebugFile" value = "2"> to write log messages only to a file in the %TEMP% folder. This approach has a drawback that only JIS log messages are written to the file and Java plugin messages are not written.
<PARAM name = "DebugFile" value = "3"> to write log messages to both the file and to the console (default).

DebugTimeStamp: When this setting is omitted from the Applet parameters, a timestamp is added by default to the file name.

DebugLevel: The existing log level 0 now provides log messages regarding errors and session dump information. A new level has been added: -1 to disable the log completely (just like debug level 0 in previous versions).

XHTML JavaScript Client Log

The XHTML client logging feature is able to log debug messages from the JavaScript used by the browser to the JIS server log.

This mechanism now has the following improvements:

- The default level is now 1 and is automatically activated (there is no longer a need to send the ClientDebugLevel URL parameter in order to activate it).
- JavaScript exceptions and their stack trace are now written to the Server Log by default.
- It is now possible to print complex messages which contain HTML text.
- It is now possible to print messages to the server log during the loading of the page.
- It is now possible to send the same message text more than once.
- The message text no longer appears in the thread name, making the text in the server log easier to read.
- Messages are written to the log in the order that they are sent from the client.

Keyboard shortcut for Java client Print GUI

The Java Client ALT+P keyboard shortcut now enables printing the active window for all windows including pop-up windows. Use the following example to customize the default keyboard shortcut:

For example, the following settings will change the Print GUI shortcut key to Ctrl+Shift+X

```
<PARAM name = "PrintGuiKeyModifier" value = "Ctrl+Shift">
<PARAM name = "PrintGuiKey" value = "X">
```

Localization Improvements

Localizing Dynamic Control Strings

JIS supports localization by means of externalizing static strings defined during design time into a resource file. The process is explained in chapter 4 of the Java client user manual.

Until now the localization feature had a limitation that only static strings (i.e. strings of components which do not have data flow) in runtime were written to the resource file (StringResource.res).

The current enhancement adds support for selectively writing dynamic strings of controls (i.e. strings of controls which have data flow) into the resource file. The dynamic strings that are to be written to the resource file are determined using selection rules. These rules define where to search and what to search for (using regular expressions). When the control name matches the regular expression defined in the selection rule, the control's string is written in the resource file.

Note that the general localization setup and procedures were not changed by this feature.

Localization of Control Strings in Design Time¶

Dynamic control strings which match one of the selection rules are written to the StringResource.res file during the runtime generation process. The strings written are the strings which appear in design time as they appear in ACE design view.

Configuring the Selection Rules:

In the Specific.ini [LocalizationExpressions] section of each library, define rules to determine which control strings will be written in the resource file. Each line in this section defines a selection rule. The structure of the rule is:

```
$Key = $Value
```

Using $Key define the sub-application name ($SubApplicationName) and control type ($ControlType). Control types can be one of the following values: All, GroupBox, Frame, TabFolder, DynamicGroup, DynamicIteration, PushButton, CheckBox, RadioButton, RadioGroup, CheckBox, OwnerDrawPushButton, PictureButton, Link, Static, Table, Edit, Window, Prompt, EditMultiline, Tabs, Menu, MenuItem or CheckboxMenuItem.

The format of the $Key token can be one of the following:

```
$SubApplicationName.$ControlType=
$SubApplicationName.All=
$ControlType=
All=
```

The format of the $Value token is a standard Perl regular expression for matching a control's name. A comprehensive introduction to Perl regular expressions can be found here: http://perldoc.perl.org/perlre.html#Regular-Expressions

Order of Evaluation:

When more than one selection rule matches a control name, the order of evaluation is as follows:

A selection rule for a specific control type in a specific sub-application takes precedence over a selection rule set for All control types in a specific sub-application.

A selection rule for All controls in a specific sub-application takes precedence over a selection rule set for a specific control type in all sub-applications.

The All definition (all control types in all sub-applications) is used if no other selection rule matches a control.

Note: If the regular expression defined in a selection rule didn't match the control name, the control's string will not be written to the resource file (i.e the less specific selection rule will not be evaluated for this control).

Note: All examples below assume the controls have data flow in ACE. Strings of controls without data flow are written to the resource file, no matter whether or not they match the selection rules.

Example 1:

The following are examples of the [LocalizationExpressions] section in the Specific.ini file:

```
[LocalizationExpressions]
LOGIN.PictureButton=^S.*
LOGIN.All=^M.*
PictureButton=^R.*
All=.*
```

In this example PictureButton control strings in the LOGIN sub-application will be written to the resource file if their name starts with the letter "S". All other controls in the LOGIN sub-application will be written to the resource file if their name starts with the letter "M". PictureButton controls in all sub-applications (other than LOGIN) will be written to the resource file if their name starts with the letter "R". All other controls (that are not PictureButton type) in all other sub-applications (other than LOGIN) will be written to the resource file regardless of their control name.

Example 2:

```
[LocalizationExpressions]
All=^(?!DDS).*
```

In this example all control strings that do not start with the prefix "DDS" will be written to the resource file and localized in runtime according to the user's locale settings.

Localization of Table Headers

Dynamic table header strings can be written to the localization resource file without defining selection rules by using the following ini setting:

```
[JAVA]
UseStaticTableHeaders=1
```

Localization of Control Strings in Runtime

Previously, only static control strings were localized in runtime according to the user locale. Now dynamic control strings, for controls that matched one the localization expressions, are also localized.

Note: The dynamic strings which are written in the resource file are only those strings that are found in the design view when generating the runtime. In runtime, these dynamic strings may change and as these values were not found previously when generating the runtime, they are not included in the resource file.

In order to add these strings to the resource file, they must be identified in runtime and recorded into the resource file manually. The identification of the strings is done by overriding the method:

ApplSubApplWindow.java

```
Public String windowMissingResource(String key) { ... }
```

The method receives the string that wasn't found in the resource file, and returns the localized version of the string. By default, the method returns the given text as is (or prefixed by "?", when working in a localization debug mode).

```
   public String windowMissingResource(String key) {
      // Write the missing string to a file or send message to the administrator
      return key;
  }
```

Limitation

This feature is not supported in the XHTML client.

Multiplying Default Control Size by a Pre-Defined Factor

When localizing strings, the translated string is often longer than the original English string. JIS previously had limited support for changing the width of the controls by a pre-defined factor, and this support has now been expanded to also support controls without data flow.

The width factor setting affects the width of the control in the following cases:

When creating a new control from a knowledgebase representation definition.

When creating a new control from a Floating representation definition.

When creating a new control using "Add Control" in design view.

When choosing "Adjust size by text" on an existing control in design view.

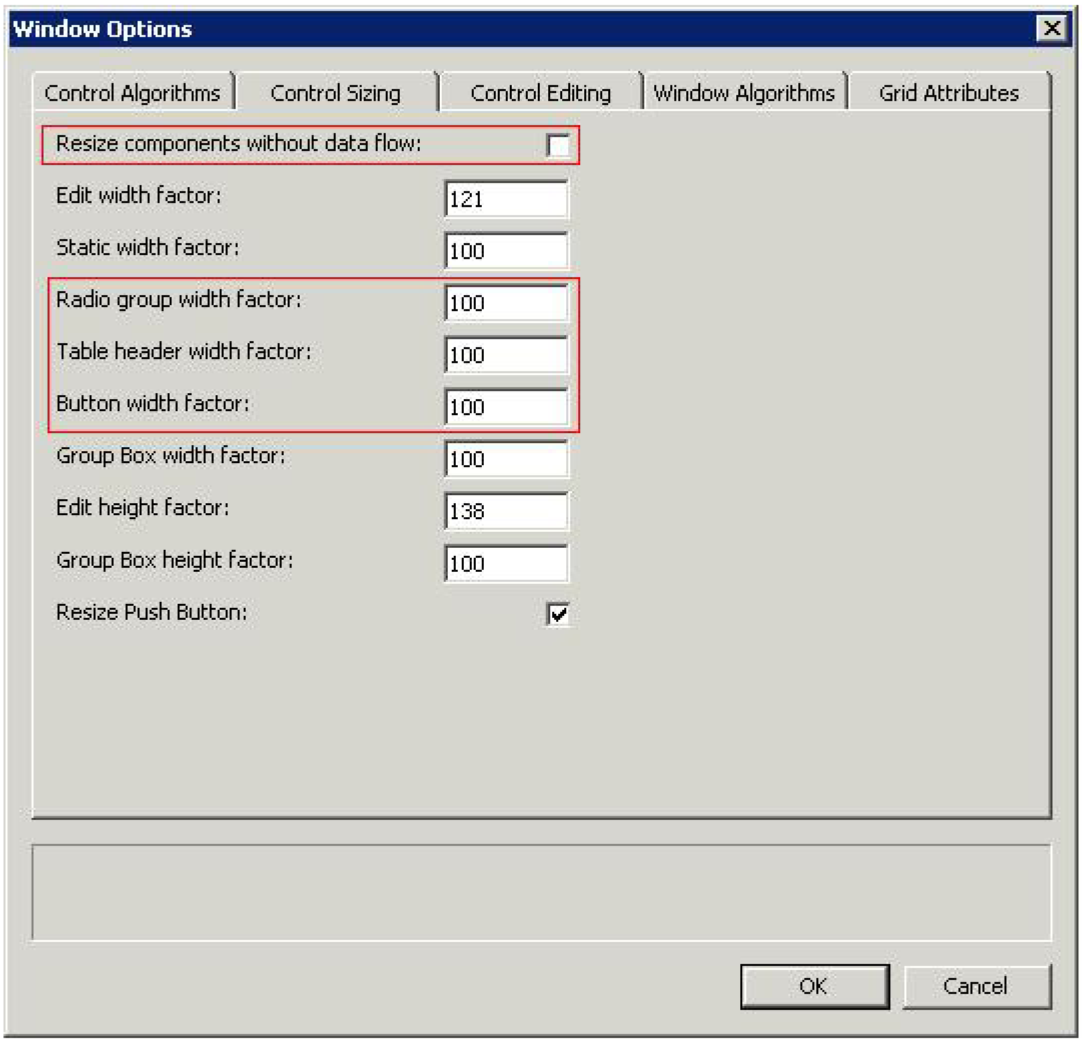
This feature changes the behavior of the various control types as follows:
Static (Static, Checkbox header) - enables sizing static controls without data flow.
Edit (Edit, Prompt, Date, Combobox, Spin) - no change, existing width factor already works.
Group box - no change, existing width factor already works.
Table headers - adjusts the table column width only if the table header multiplied by the width factor is wider than the table column data area. The assumption is that only table headers will be localized, while data displayed within the table data area will not be localized.
Button (PictureButton, PushButton, Link) - width factor is calculated based on the current button text.
Radio group - in order to determine the default component size, the width factor is multiplied by the longest Radio item.
Tab header – no change. Tab headers already have a setting similar to width factor named:

```
[Converter]
AddTabX=
```

The AddTabX setting can be used to increase the width of the tab header by a given number of pixels.

The width factor settings can be configured in the Window Options dialog box, in a new tab named “Control Sizing”. The “Control Sizing” tab includes previously supported control sizing parameters as well as the newly supported parameters. The new parameters include: radio group width, table header width and button width.

The new setting "Resize components without data flow" was introduced in order to maintain backward compatibility. By default, the checkbox is unchecked so that components which have no data flow will not be resized. Select this checkbox so that the various width factors will also affect the controls which have no data flow.


Window Options

Resize components without data flow:	☐
Edit width factor:	121
Static width factor:	100
Radio group width factor:	100
Table header width factor:	100
Button width factor:	100
Group Box width factor:	100
Edit height factor:	138
Group Box height factor:	100
Resize Push Button:	✓

Limitations

The width factor settings only affects components which has no local modifications (manual or using window layout).

The "Adjust Size By Text" function does not resize table headers.

After making changes to the "Window Options" dialog the user needs to click the <Apply> button in order for the changes to take effect.

New Features in Service Pack 9.0.4

Logging Improvements

Logging functionality has been improved:

- The architecture parameter 32bit or 64bit is now written to the log.
- The current time zone is now written to the log.
- The time stamp written to the log now includes milliseconds.
- The AM/PM marker has been removed and replaced with 24 hours time.

Simplifying JIS Windows Service Configuration

Deploying JIS as a Windows service has been simplified. When running the JBSToService.exe utility after creating a JIS runtime installation folder for Windows (do not run JBSToService.exe from the JIS installation folder itself), most of the default values are calculated correctly as follows:

a. The code is able to automatically locate JBSService.exe.

b. The code cleans up and uses the command line from the existing jacadasv.bat when launching the service (there's no longer a need to clean up the % signs).

c. The default ini settings are read from the jacadasv.ini of the runtime installation (note that by default there's no need to specify the settings).

d. The service log is automatically created in ..\classes\logs\JBSService.log.

In addition, the log messages have been improved and time stamps have been added.

Backward compatibility: The new implementation maintains backward compatibility with existing JBSToService command line options.

For example for an application named XHTMLV9:

1. Create the service: C:\XHTMLV9\bin>JBSToService.exe –c
   Service name: JISSvc
   Display name: JIS Service
   Description: Controls the running of a JIS Server
   Path to executable: C:\XHTMLV9\bin\JBSService.exe
2. Service “JIS Service” now appears in the services control panel. You can start and stop it using the standard services panel.
   Remove the service:
   C:\XHTMLV9\bin>JBSToService.exe -r

Using JAM as an Applet in JIS Standalone Server

JAM can now run as an Applet also when using the standalone server. The main advantage of this configuration is that it does not require opening any ports in the Firewall. In previous versions when running the JIS server on Unix, users had to either use an X-terminal for running JAM or open several ports in the Firewall in order to run JAM from a Windows workstation. This is no longer necessary.

To access the JAM Applet from the development environment use the following URL: http://localhost:8080/JISAdminServlet. In production configuration replace localhost:8080 with your server address and port.

When running as an Applet, JAM is password protected. The default username/password is: jisadmin/jisadmin.

Secure Login to JISAdminServlet

When JAM is running as an applet, the login to JAM is secured and requires a username and password (required when accessing http://<host>:<port>/JISAdminServlet). The username and password can be specified in the jacadasv.ini file, under the [HTTP] section, using the JAMUsername and JAMPassword keywords. The value of the JAMPassword can be written as an encrypted password. Generate the encrypted password using the batch file located in <JIS installation folder>\JacadaFiles \utils\web\jetty\encodePassword.bat. If the JAMUsername and JAMPassword are not specified in the ini file, jisadmin is used for both the username and password.

Note: When accessing JAM via Internet Explorer, you are required to enter your user name and password twice.

Example for generating an encrypted password:

1. From a command prompt, execute:

```
C:\XHTMLV9\utils\web\jetty>encodePassword.bat mypass
…
OBF:1xfd1zt11uha1ugg1zsp1xfp
MD5:a029d0df84eb5549c641e04a9ef389e5
```

2. Add the following setting to jacadasv.ini:

```
[HTTP]
JAMUsername=myuser
JAMPassword=OBF:1xfd1zt11uha1ugg1zsp1xfp
```

JIS Administrator Command Line Operations

The standalone version of JAM now provides command line interface for performing operations such as shutting down the server, suspending connections of new users, resuming activity on the server and checking the status of the server.

To use the command line interface, open a command prompt, navigate to the <JISRoot> folder and issue a JAM –x <command> as shown below.

```
jam -x  shutdown <time in minutes>
```

Closes a JIS server after a time interval specified in minutes (when the time interval is not specified, the server is closed immediately).

```
jam -x  suspend
```

Suspends connections of new users to the JIS server.

```
jam -x  resume
```

Resumes activity on the JIS server.

```
jam -x  status
```

Checks if the JIS server is running.

The “status” command has the following return codes:

Code	Description
1	The server is running
-1	The server is not running or there’s a communication problem between JAM and the server.

In order to check the value of the status command, you can create the following CheckServerStatus.bat file in your <JISRoot> folder:

```
@echo off
call jam -x status
IF %ERRORLEVEL% EQU -1 goto servererror
echo CheckServerStatus: server is Ok
goto exit

:servererror
echo CheckServerStatus: something is wrong with the server or with the connection from jam to
the server

:exit
```

Modifications made to the J2EE Deployment Procedure (XHTML only)

The JIS common installation for J2EE will no longer attempt to update the classpath of an application server or deploy the application EAR files automatically.

After installing the common installation and before deploying the application ear file, add the jar files placed in the common installation \lib folder into the application server's classpath.

As of JIS 9.0.4 the jar files are:

- jacadasv.jar
- Tidy.jar
- sac.jar
- cssparser-0.9.5.jar

Adding jar files to an application server's classpath is an application server specific procedure. Please consult your application server documentation.

Specifically, adding jar files to the WebSphere application server is documented in the XHTML user guide.

Upgrade to Jetty 6.1

The embedded Jetty web server bundled with JIS has been upgraded to version 6.1.19. In addition, a few more configurations have been introduced:

1. The ability to use HTTPS only and disable the HTTP port. Use the following jacadasv.ini setting: [HTTP] SupportHttpsOnly=1
2. The ability to disable directory browsing which is on by default. Use the following jacadasv.ini setting: [HTTP] AllowDirectoryBrowsing=0
3. The ability to hide some of the server resources from the client. [HTTP] ProtectedResources=/classes/MyFile1.txt,/classes/MyClass.class

   When trying to access these resources from a URL such as: http://myserver:8080/classes/MyFile1.txt the client will receive 404 response. The following resources are protected by default:

   /classes/http.xml

   /classes/jetty-jmx.xml

   /classes/jacadasv.ini

   /classes/jacadasv.policy

   /classes/jcedit.res

   /classes/jrodefaults.ini

   /classes/license.dat

   /classes/proxyConfiguration.xml

   /classes/proxyHttp.xml

   /classes/ServerConfiguration.dtd

   /classes/ServerConfiguration.xml

   /classes/JettyKeyStore

   /classes/cst/jacadasv.jar

4. The ResourceBase property now defaults to the RtRootDir property.

Backward compatibility:

In this release action definition names specified in ServerConfiguration.xml such as "Xhtml" or "FreeSession" are case sensitive (they were case insensitive until now). JIS recognizes action names as they are written in ServerConfiguration.xml or as all upper case or as all lower case. For example, the action /FreeSession can also be used as /FREESESSION or /freesession.

Running the JacadaProxyServlet as part of the JIS Server

The Java client can now communicate directly with the JIS server using HTTP/S without having to deploy the JacadaProxyServlet to an external Servlet engine. The Servlet is now run using the embedded Jetty 6.1 servlet engine.

To configure a client to connect to the server using HTTP, add the following Applet parameters:

<PARAM name = "UseHttp" value = "true">

<PARAM name = "UsePorts" value = "false"> .

In addition, a new .html page (<ApplName>-JavaClientHttp.html) is generated during Generate Runtime. The page contains the necessary definitions for the Java client to connect to the server from which it was downloaded via HTTP/S.

The embedded ProxyServlet always works in non persistent mode. The request used for sending messages from the server to the client is closed by the server and opened by the client after specific protocol messages. This ensures that the client does not keep an open connection to the server for long periods of time.

Configuration: When using the embedded ProxyServlet, the following jacadasv.ini settings replace settings which were configurable in the web.xml when deploying the ProxyServlet as a standalone component:

[JISProxyServlet]

HideException - hide exceptions thrown by the ProxyServlet from the client [default: 0].

EnableTestServlets - enables the test servlets for researching communication problems [default: 0]

GetClientIPFromHttpHeader - allows to retrieve the client IP from an HTTP header [default: 0]

ClientIPHttpHeaderName - the name of the HTTP header from which to read the client address [default is empty]

Logging: The embedded ProxyServlet writes log messages to the standard server log. There are no longer jac-<sessionid>.log files.

Backward compatibility:

You can still package the standalone JacadaProxyServlet classes and deploy them to your desired servlet engine. However it is recommended to start planning their migration to the embedded proxy servlet.

The HTTP communication mode is optional. You can still use the standard ports communication.

The HttpDebugLevel applet parameter is now obsolete.

Reduction of the size of the XHTML file.

This feature reduces the HTML page size generated by JIS and in this way reduces the network bandwidth consumed by JIS applications. In addition it also accelerates the generation of the XHTML page in runtime:

- Position related style attributes of HTML elements are no longer part of the page itself, instead they are externalized into a CSS. The CSS is generated in runtime the first time the sub application window is accessed.
- The generated CSS for a sub-application is sent to the browser once per session.

We predict that this will reduce the page size generated by JIS by approximately 30%.

The feature is enabled by default and can be disabled by setting the following parameter to 0.

```
[XHTML]
OptimizeStyleAttributes=0 (1 is the default value)
```

Allowing the User to Adjust the Java Client Debug Level

It is now possible for the end user to set the debug level of the Java client logs in the current session. This can be done by selecting Application>File>Adjust Debug Level or by clicking on a key combination (defined by the JIS developer). The key combination can be defined in the DebugLevelAdjustKey (set to any valid single character) and DebugLevelAdjustKeyModifier (set to Ctrl, alt or Shift) applet parameters inside the HTML page. When not specified, the default key combination is ALT+d.

This feature is useful for debugging resource problems in the Java client. The user can start the session with debug level 1 and only increase the debug level when a problem such as slowdown is observed.

Post Class Path

This allows running the JIS server with an additional set of jars. It is possible to add an extra token to the classpath, which is appended at the end of the default JIS classpath. This is done in the jacadasv.ini file, where you can set, within the [VMCommandLine] section, the PostClasspath setting to list all the required jar files, which are not part of the JIS default classpath.

Example:

```
[VMCommandLine]
PostClasspath=c:\jdbc\jdbc.jar;myapp.jar
```

New Methods for Handling User Variables

The following DoMethods were introduced:

```
DoMethod: Receiver: System Method: logSharedUserVariables Parms: ( <debug level> )
DoMethod: Receiver: System Method: logUserVariables Parms: ( <debug level> )
```

The following public APIs were introduced:

```
/**
Retrieve all user variables
@return Map of variables in key,value pair format.
*/
public Map getUserVariables();
```

```
/**
print to the server log file all user variables
@param debugLevel variables will be printed when the server log debug level is equal or higher
than debugLevel
*/
public void logUserVariables(int debugLevel);
```

```
/**
Retrieve all shared user variables
@return Map of shared variables in key,value pair format.
*/
public Map getSharedUserVariables();
```

```
/**
print to the server log file all shared user variables
@param debugLevel variables will be printed when the server log debug level is equal or higher
than debugLevel
*/
public void logSharedUserVariables(int debugLevel);
```

Usage examples:

1. From within an ACE method:

```
Action: Enter
Trigger: 18000 WaitIndicator: True ScrambleName: False MoveMode: MoveNone Description: This
method presses the Enter key on the host, and then proceeds to the next screen.
Update: Fields: ( _All_Fields_ ) From: TheWindow To: TheScreen
DoMethod: Receiver: System Method: logSharedUserVariables Parms: ( 50 )
DoMethod: Receiver: System Method: logUserVariables Parms: ( 70 )
DoMethod: Receiver: SubApplication Method: SetCursorPosOnScreenAccordingToFocusedControl Parms:
( )
HostType: AidKey: AidEnter RemainInScreen: False
DoMethod: Receiver: SubApplication Method: MoveAccordingToHost Parms: ( )
```

2. From within a server side extension:

```
package appls.TESTB48.server.user;

import cst.server.general.*;
import java.util.*;
```

```
public class GeneralSubApplication extends appls.TESTB48.server.original.GeneralInternalSubAppl
{
      public GeneralSubApplication (Globals globals_parm){
             super (globals_parm);
             return ;
      }
      public void u_Enter(int lParam) {
             super.u_Enter(lParam);
             Map vars = globals.system().getSharedUserVariables();
             if (vars != null) {
                    Set keys = vars.keySet();
                    Iterator iterator = keys.iterator();
                    while (iterator.hasNext()) {
                           String key = (String)iterator.next();
                           String value = (String)vars.get(key);
                           // do something useful
                    }
             }
      }
}
```

Rebranding

The software and product documentation has been rebranded to suit Software AG standards.

New Features in Service Pack 9.0.3

Changes in the Product Name

As a result of the acquisition of the Jacada application modernization product line by Software AG, we have begun the process of updating the product name to suit the company standards. The product name is now JIS, and we have begun to implement this throughout the product. This has not yet been implemented in the tutorials and in the documentation.

Runtime Installation Improvements

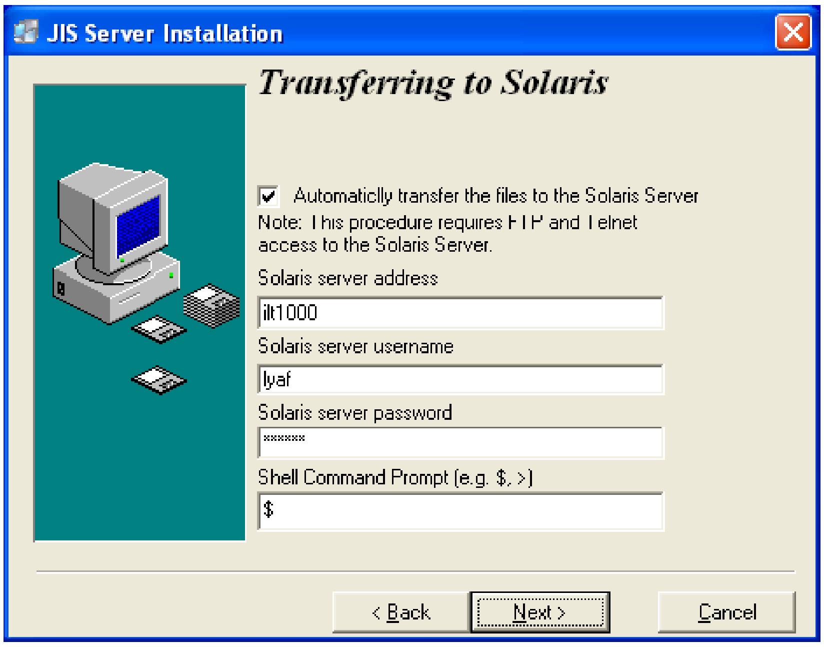
The runtime installation process now enables installing and deploying JIS automatically on UNIX platforms. At the end of the Wise installation process, an ANT script will be invoked to transfer the files to the UNIX machine and configure the JIS server. Two optional, new dialog boxes in the installation process enable implementing this feature. Refer to the runtime installation process in the documentation for details.

UNIX Machine Prerequisites:
FTP access enabled.
Telnet access enabled.

Unzip command installed.

Changes in the installation process:

In the following screen, when selecting to automatically transfer the installation to UNIX by FTP, you will be required to enter the IP address or name of the UNIX machine where the JIS server is to be installed, the name and password of a user who has permissions on the machine to connect using FTP and Telnet and the postfix of the shell command prompt string to be used by the UNIX machine.

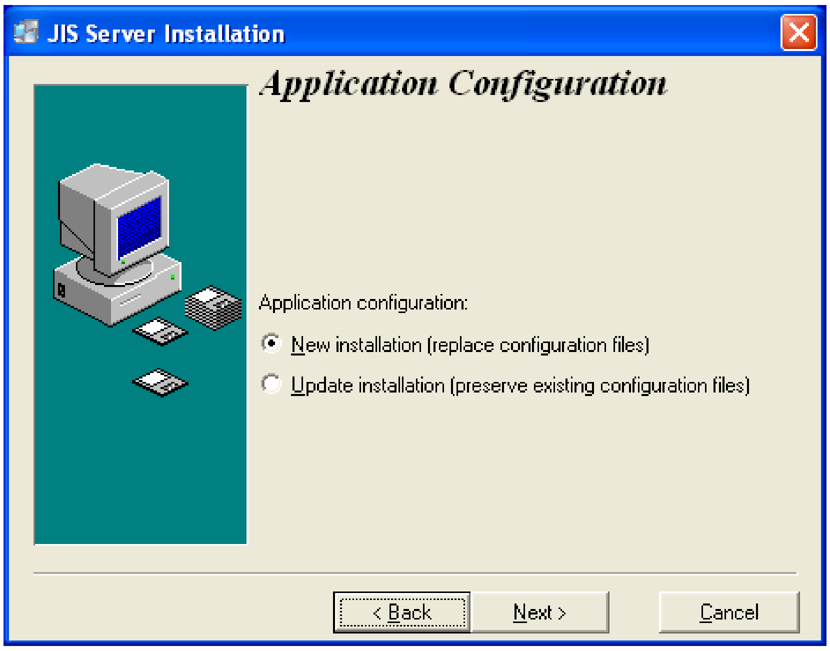

JIS Server Installation

Transferring to Solaris

[✓] Automaticlly transfer the files to the Solaris Server
Note: This procedure requires FTP and Telnet access to the Solaris Server.

Solaris server address: ilt1000
Solaris server username: lyaf
Solaris server password: *******
Shell Command Prompt (e.g. $, >): $

The screen which follows the above screen is the Application Configuration screen, where you are required to select whether to replace the configuration files or to preserve existing configuration files.


JIS Server Installation

Application Configuration

Application configuration:
[● New installation (replace configuration files)]
Update installation (preserve existing configuration files)

Note: The new installation procedure is only available for Solaris, AIX and Linux.

Simplifying the Printing Emulation Configuration (XHTML)

The printing emulation configuration for JIS XHTML has now been simplified, and default values are provided for most parameters.

The following parameters now have defaults which are suitable for most configurations and no longer need to be defined in the <ApplicationName>.ini:

```
[TN5250 Printer]
WorkRootURL
WorkRootDirectory
SpoolDirectory
XSLTforXMLtoHTML
[Printing Handlers] section
```

The following parameters still need to be configured in order to enable printing:

```
[GUISys TN5250]
; enable host printing
Printer=1
[TN5250 Printer]
; set the device name
LUName=<Device name>
```

Improved Host Language Support

Introduction

JIS support of host languages has been simplified. JIS has now integrated the "descriptor" mechanism, which enables support of more than one host language on the same server requiring minimal configuration. Users no longer need to use the complex and error prone LanguageDescriptorFactory extensions. This version of JIS also clarifies the level of support to languages that were not previously supported.

An additional language related enhancement which has been added supports printing special characters to the log file, improving debugging capabilities.

Note: Language Descriptors can still be customized at the project level to maintain backward compatibility.

Using this Feature

The following details how to enable and configure this feature on the server, the Java Client and the XHTML client.

Server:

Add the following <ApplicationName>.ini setting to enable the feature:

```
[Emulator]
LanguageDescriptorEnabled=1
```

Note: Applications that have used `LanguageDescriptorFactory` extensions in previous versions will also need to add this setting when upgrading.

Java Client:

1. Specify the host language used by the client using the following Applet parameter: `<PARAM name = "LanguageDescriptor" value = "<Language Name>">`.
2. Make sure the client operating system supports the language specified in its "Regional Settings" and that the language specific fonts, if there are any, are installed.
3. When using Chinese, Japanese, Korean or Thai, add the clcharsets.jar file to the ARCHIVE tag and the clcharsets.cab file to the Cabbase parameter in the launcher Html page. If you are using one of these languages and the clcharsets archive is not added this will cause the LanguageNotSupported.html page to be presented when starting a session.
4. When using some languages (such as Japanese, Chinese or Thai) the default fonts used by the JIS host screen and the JIS JITGUI sub-application do not display the screen contents correctly. Instead you may see square signs or question marks. To fix this experiment with the following Applet parameters:
   `<PARAM name = "CourierFontType"      value = "Courier">` to control the font in the JITGUI and dynamic areas.
   `<PARAM name = "EmulatorFontName"      value = "Courier">` to control the font used by the Host Screen.

XHTML Client:

1. Specify the Language used by the client using the following URL parameter or post data or Http header "LanguageDescriptor=<Language Name>". Note that you cannot pass this parameter to the <AppName>-xhtml.html. You either have to code it in the <AppName>-xhtml.html itself or write the full URL in the browser's address bar such as /XHTML?JacadaApp.
2. Make sure the client operating system supports the language specified in its "Regional Settings" and that the language specific fonts, if any, are installed.

Customization

It is still possible to customize the LanguageDescriptor provided by the product for project specific requirements. In order to do this, the language descriptor classes must be named as follows:

Client side: `appls.<AppName>.user.User<LanguageName>LanguageDescriptor`

Server side: `appls.<AppName>.server.user.User<LanguageName>LanguageDescriptor`

Log Files

The server log and Java client log now displays field content encoded using the current session language encoding. To view the log files with the correct encoding, we recommend viewing the log file from a client machine which supports the encoding used by the sessions and use an encoding aware text editor such as Wordpad.

Backwards Compatibilility

Applications which do not use LanguageDescriptors

Existing applications that use the application level language setting in ACE, and do not use language descriptors, will continue working as before.

Applications which use LanguageDescriptor extensions

Existing applications already using language Descriptors implemented as extensions should first try to use the internal descriptors and remove and discontinue the usage of the extensions.

Only if you must continue using the extensions, then set the following:

In the <ApplicationName>.ini file:

```
[Emulator]
LanguageDescriptorEnabled=1
LanguageDescriptorFactory=appls.<AppName>.server.user. ILanguageDescriptorFactory
```

In the HTML Launcher:

```
<PARAM name = "LanguageDescriptorFactory" value = "
appls.<AppName>.user.ILanguageDescriptorFactory">
Where <AppName> is the name of the JIS application.
<PARAM name = "LanguageDescriptor" value = "<Language name from the Supported Languages table">
```

In addition if you are using an existing ChineseLanguageDescriptor class on the server and client, you'll need to:

1. Change the isLanguageSupported () method as follows:

```
   public boolean isLanguageSupported(String language) {
       return language.equalsIgnoreCase("Chinese");
   }
```

2. In the ILanguageDescriptorFactory class, replace occurrences of "Chinese (Simplified)" with "Chinese".

Parameters:

This section provides a reference to the configurable parameters:

LanguageDescriptorEnabled

Enables the "Descriptor" mechanism within JIS. Once this parameter is set, both the internal language descriptors and language descriptor extensions are enabled.

Configuration file: <ApplicationName>.ini.

Section: [Emulator]

Possible values: 0, 1 (default - 0).

Example: LanguageDescriptorEnabled=1

LanguageDescriptorFactory ini setting & Applet parameter

Allows using a server side LanguageDescriptorFactory for backward compatibility. Users upgrading their descriptors from an earlier version, who would like to continue to use the external descriptors, should configure this setting.

Configuration file: <ApplicationName>.ini

Section within file: [Emulator]

Possible values: Class name (default - uses internal factory.)

Example:

ini setting:

```
LanguageDescriptorFactory=appls.MYAPP.server.user.ILanguageDescriptorFactory
```

Applet parameter:

```
<PARAM name ="LanguageDescriptorFactory" value="appls.MYAPP.user.ILanguageDescriptorFactory">
```

LanguageDescriptor Applet parameter & URL parameter

Name of the language descriptor to be used by this client session.

Configuration file: Java client launcher HTML. Section: Applet parameters

Possible values: Are listed in the Supported Languages table.

Example:

Applet parameter:

```
<PARAM name ="LanguageDescriptor" value="Chinese">
```

URL parameter:

```
http://localhost:8080/Xhtml?JacadaApplicationName=MYAPP&LanguageDescriptor=Chinese
```

Parameters for Backwards Compatible Settings (to be used only when not using the "Descriptor" mechanism)

Conversion File

This value will override the host code page defined by the language descriptor for all sessions.

Configuration file: <ApplicationName>.ini. Section: [GUISys TN5250]

HostCodePage

This value will override the host code page defined by the language descriptor for all sessions.

Configuration file: <ApplicationName>.ini. Section: [GUISys TN3270]

RuntimeLanguage

There is no longer a need to specify a specific server side language since the server now supports all languages

Configuration file: <ApplicationName>.ini. Section: [Emulator]

Supported Languages

JIS Language Descriptor to codepage mapping

Language Descriptor	Class Name Prefix	EBCDIC Codepage	ASCII Codepage	Is Double Byte
Default	N/A	Cp037	Cp1252	
Albanian	Albanian	Cp870	Cp1250	
Belorussian	Belorussian	Cp1025	Cp1251	
Bulgarian	Bulgarian	Cp1025	Cp1251	
Chinese (Simplified)	Chinese	GB935	GBK	Yes
Chinese (Traditional)	ChineseTraditional	Cp937	Big5	Yes
Croatian	Croatian	Cp870	Cp1250	
Czech	Czech	Cp870	Cp1250	
Danish	Danish	Cp1142	Cp1252	
English UK	EnglishUK	Cp1146	Cp1252	
English USA	EnglishUS	Cp1140	Cp1252	
French	French	Cp1147	Cp1252	
German	German	Cp1141	Cp1252	
Greek	Greek	Cp875	Cp1253	
Hungarian	Hungarian	Cp870	Cp1250	
Italian	Italian	Cp1144	Cp1252	
Japanese	Japanese	SJIS	MS932	Yes
Korean	Korean	Cp933	Cp949	Yes
Macedonian	Macedonian	Cp1025	Cp1251	
Norwegian	Norwegian	Cp1142	Cp1252	
Polish	Polish	Cp870	Cp1250	
Portuguese	Portuguese	Cp037	Cp1252	
Romanian	Romanian	Cp870	Cp1250	
Russian	Russian	Cp1025	Cp1251	
Serbian	Serbian	Cp1025	Cp1251	
Slovak	Slovak	Cp870	Cp1250	
Slovenian	Slovenian	Cp870	Cp1250	
Spanish	Spanish	Cp1145	Cp1252	
Swedish	Swedish	Cp1143	Cp1252	
Swiss-German	SwissGerman	Cp500	Cp1252	
Thai	Thai	Cp838	MS874	
Turkish	Turkish	Cp1026	Cp1254	
Ukrainian	Ukrainian	Cp1025	Cp1251	

Comments:

- The EBCDIC Codepage is being used by the server when converting information sent and received from the host into an ASCII encoding.
- The ASCII Codepage is being used by the server when converting ASCII encoded bytes into Java characters encoded using Unicode.
- All information sent and received between the clients and the server is encoded using Unicode encoding.
- For extension developers, to obtain the descriptor class name from the “Class Name Prefix” append the string “LanguageDescriptor” to the prefix.
- “Belorussian” also refers to “Belarussian” and “Slovenian” also refers to “Slovene”.

- The matching AS/400 CCSID can be obtained by looking up the “EBCDIC Codepage” for a specific language descriptor in the following link: http://publib.boulder.ibm.com/infocenter/iseries/v6r1m0/index.jsp?topic=/rzaha/fileenc.htm

- Follow this link for a list of codepages supported by Java: http://download.oracle.com/javase/6/docs/technotes/guides/intl/encoding.doc.html

Note: Customers can implement their own descriptors by extending the existing product descriptors.

Example:

```
package appls.IT.server.user;
public class UserItalianLanguageDescriptor extends ItalianLanguageDescriptor {
   // override here methods from the ILanguageDescriptor interface, for example:
   public boolean isDBCSLanguage() {
      return super.isDBCSLanguage();
   }
}
```

Recommendations: When using LanguageDescriptors, it is recommended to run the interface server with the default file encoding for the operating system and not use a specific encoding.

JIS clients were tested on a standard Windows XP SP2 operating system version (not a language specific operating system) with the specified languages enabled in the regional settings.

Java Client "About" Dialog Box

About dialog no longer contains the now obsolete RTCP key information "Serial No:" and "Licensed To:" labels.

"Host Print Transform" Printing using Java Services

When using the AS400 HPT (Host Print Transform) feature, the print job is sent from the AS400 already formatted with all the necessary escape codes required for the print job formatting. In previous releases of JIS the Java client sent this print job into a predefined parallel or serial port on the local PC. The end user had to define port capturing on the local PC for the actual printer.

This feature adds the ability to use the Java print service APIs to implement the same behavior. This way the user does not have to configure the port in advance. Java takes care of this for you. In addition HPT, using the print service, now supports the existing "PrinterEmulationPageOrientation" and "PrinterEmulationPaperType" Applet parameters.

Exposing the XHTML Page DOM for Java Extensions

This feature exposes thepage DOM for project specific extensions from the OnPageLoad() event handler of Appl.java.

A new OnPageLoadContext API:

```
public Document getXhtmlDom()
```

This should be used to retrieve the existing page DOM and change it for project specific requirements.

The DOM retrieved by getXhtmlDom() does not reflect style changes made by the existing, project specific, XHTML extensions.

In order to reflect changes made by existing XHTML extensions you need to add the following code to the onPageLoad() method:

```
public void onPageLoad(OnPageLoadContext context) {
    // Existing Xhtml extensions
    ...
    // Update the new style settings from the Xhtml extensions into the Page DOM
    Hashtable styleHash = context.getDataBlock().getStyleHash();
    for(Enumeration e = styleHash.keys();e.hasMoreElements();) {
         Element key = (Element)e.nextElement();
         StyleModifier.updateStyleAttribute(key, (Map)styleHash.get(key));
    }
    ...
    // get the already modified DOM and further manipulate it
    Document xhtmlDom = context.getXhtmlDom(); ...
}
```

Another important note is that while you are modifying the component style inside the DOM object, you have to clone the modified component, see example (otherwise the JIS internal code will re-apply the styles set in the code extension and override the DOM manipulations.).

Code example:

```
 package appls.XHTMLV9.xhtml.user;
 import com.jacada.jis.runtime.server.frontend.xhtml.context.*;
 import com.jacada.jis.runtime.server.frontend.xhtml.general.XhtmlConstants;
 import com.jacada.jis.runtime.server.frontend.xhtml.controls.XhtmlControl;
 import org.w3c.dom.*;
 import java.util.*;
 import com.jacada.jis.runtime.server.frontend.xhtml.controls.Window;
 import com.jacada.jis.runtime.server.frontend.xhtml.modifier.StyleModifier;
 /**
  * description : Appl.java
  */
 public class Appl implements
com.jacada.jis.runtime.server.frontend.xhtml.extension.IUserPageExtension {
    /**
     * Constructor
     */
    public Appl () {
    }
    public void onPageLoad(OnPageLoadContext context) {
       Window window = context.getWindow();
       Vector vControls = window.getAllControls();
       for(int i = 0; i < vControls.size(); i++) {
          // Get the control on the window
          XhtmlControl curControl = (XhtmlControl) vControls.get(i);
          curControl.setForeground("RED");
          //curControl.setSize(100,10);
          //curControl.setLocation(1,1);
       }

       // Update the new style settings into the Page DOM
       Hashtable styleHash = context.getDataBlock().getStyleHash();
       for(Enumeration e = styleHash.keys();e.hasMoreElements();) {
        Element key = (Element)e.nextElement();
        StyleModifier.updateStyleAttribute(key, (Map)styleHash.get(key));
       }

      //this is an example of manipulating label nodes on a JITGUI screen
      //converted screens have different structure specially if they
        contain tab controls
      Document xhtmlDom = context.getXhtmlDom();
      NodeList formList = xhtmlDom.getElementsByTagName(XhtmlConstants.HTML_FORM);
       Node form = formList.item(0);  // we assume this is the jacadaform
       but you need to check to make sure
       NodeList formElements = form.getChildNodes();
       for (int i=0; i < formElements.getLength(); i++) {
          Node formNode = formElements.item(i);
          if ("span".equalsIgnoreCase(formNode.getNodeName())) {
             NodeList spanChildNodes = formNode.getChildNodes();
             for (int j =0; j < spanChildNodes.getLength(); j++) {
                Node spanChildNode = spanChildNodes.item(j);
                if ("pre".equalsIgnoreCase(spanChildNode.getNodeName())) {
                  NodeList preChildNodes = spanChildNode.getChildNodes();
                  for (int k =0; k < preChildNodes.getLength(); k++) {
                   Node preChildNode = preChildNodes.item(k);
                   if("label".equalsIgnoreCase(preChildNode.getNodeName()))
{
                   // clone the label, change its color to Blue and update it back to the dom
                    Element label = (Element)preChildNode.cloneNode(true);
                    String style = label.getAttribute("style");
                    style = style.replaceFirst("color:RED", "color:#0000ff");
                         label.setAttribute("style", style);
                         spanChildNode.replaceChild(label, preChildNode);
                      }
                   }
                }
             }
          }
       }
    }

    public void onPageSubmit(OnPageSubmitContext onSubmitContext) {

    }
```

```
}
```

AutoSkip Supported in XHTML

AutoSkip is now supported enabling automatically skipping to the next field in the tab order, once the field has been filled and the caret is at the end of this field.

In the <ApplicationName>.ini file, in the [XHTML] section, configure the `AutoSkipSupport` parameter:

`AutoSkipSupport=0` does not support using autoskip.

`AutoSkipSupport=1` supports using autoskip (default value).

New Features in Service Pack 9.0B

Command-Line Access to ACE

Command-line access to ACE enables automating the following operations:

- Generate Runtime
- Create Runtime Installation
- Pack/Unpack

The command-line access mechanism allows external tools to run ACE in automatic mode, using an XML file that contains a list of operations to execute in ACE.

The XML file must be named buildapp.xml.

Place the buildapp.xml file in the folder from which you launch ACE (the folder containing the ACE executable).

To launch ACE in automatic mode:

- Use the following command parameter: -oREMOTE
  For example, to launch JIS XHTML for 3270 in automatic mode, use the following command line: ACE.EXE -lmp -oREMOTE

When running ACE with the -oREMOTE command-line parameter, the buildapp.xml is read and the operations are executed by ACE.

Running ACE in Automatic Mode

When ACE is launched in automatic mode, it creates a GUI. Additional windows, such as the Pack animation window, are also displayed. However, in automatic mode, the ACE GUI is disabled.

When all of the operations run successfully (without errors), no user intervention is required. So, for instance, the Generate Runtime process dialog still opens, but closes automatically when it is done. If, however, there is an error message, such as an alert about missing image files, then this message is shown, and the user needs to click *OK* to close it. For more information, see “ErrorHandling”.

Once a valid buildapp.xml file has been created and tested, the operations run in ACE without the need for user intervention.

Changes in the Behavior of ACE Operations

The following sections describe changes in the behavior of ACE operations when using automatic mode.

Changes in Generate Runtime

The Generate Runtime process is always carried out for the entire application. It is not possible to specify specific libraries or subapplications. To speed things up, you can use the option to compile only new and modified subapplications.

Changes in Create Runtime Installation with Wise

After creating a runtime installation with Wise, ACE asks whether to launch the newly-created installation. In automatic mode, this question is skipped, and the installation is not launched.

Changes in Pack

When packing an application, it is possible to select the libraries to pack, and to add additional files, but the other steps of the Pack Wizard are not supported. Thus, for example, it is not possible to specify a maximal file size, nor to skip input directories (such as skipping DDS files, installation files and configuration files). All the files are included in the package (including configuration files).

Changes in Unpack

When unpacking an application, existing files are automatically replaced.

The configuration files (the files asked about in the last step of the wizard) are unpacked according to the settings chosen the last time the wizard was run from ACE.

Example - buildapp.xml

The following example demonstrates various operations, such as opening an application, generating a runtime, creating a runtime installation, and packing and unpacking an application:

```
<Ace>
 <OpenApplication Name="TEST1">
  <GenerateRuntime Type="Java;XHTML" Platform="Windows; Solaris;
                   OS390;AS400;AIX;Linux" NewAndModified="0" />
 </OpenApplication>

 <OpenApplication Name="TEST2">
  <GenerateRuntime Type="Java;XHTML"
            Platform="Windows;Solaris;Linux;AIX;AS400;OS390" />
  <CreateRuntimeInstallation DeploymentType="Standalone"
     Platform="Windows" Runtime="XHTML" InstallFileSize="1024">
   <Wise Launch="1" ExecuteFile="C:\Program Files\Wise
         InstallMaster Demo\wise32.exe"
         InstallationDirectory="C:\TEST2" ImageFile="">
   </Wise>
  </CreateRuntimeInstallation>

  <CreateRuntimeInstallation DeploymentType="J2EE"
                            Server="weblogic;tomcat;websphere">
   <Libraries List="MODELS"/>
   <AdditionalFiles>
    <File Name="c:\temp\debug_1.log" Target="\WEB-INF\Lib" />
    <File Name="text2" Target="\WEB-INF\classes" />
   </AdditionalFiles>
  </CreateRuntimeInstallation>
 </OpenApplication>

 <Pack File="c:\temp\packtest.jpk" Name="TEST2" Libraries= ";"
                                         AdditionalFiles=";" />
 <Unpack File="c:\temp\myapp.jpk" Type="Java"
         Target="MYAPP"
         InputDirectories="DDS;SDF;Screens"
         OutputDirectories="MakeExe;Runtime;Install"
         IncludeExtraFiles="True"  Existing="Replace"
         ConfigurationFiles="All">
 </Unpack>
</Ace>
```

For the complete DTD, which describes all possible attributes, refer to the Buildapp.DTD.

Logging

When running ACE in automatic mode, the following log is created: remote.log in the ACE root folder. The log begins with the contents of the buildapp.xml file. The remote.log reports the progress of the operations listed in the buildapp.xml file. In addition, information is also logged to the standard logs created by the Generate Runtime and Pack/Unpack Wizards.

Error Handling

When there is an an error in any of the operations in the file, processing will terminate immediately. This prevents the accumulation of several problems, one on top of the other. For instance, if generating the runtime fails, then creating a runtime installation might create an installation of the previous version.

The following table describes different types of errors that may occur, and how to handle them.

Error Type	**Reasons for Error**	**What to Check**
Syntax errors in buildapp.xml	• Malformed XML (i.e. XML tags not closed) • Children tags appear outside of parent tags • Missing values in XML, missing attributes	• Verify the syntax of the XML • Verify that the XML conforms to the DTD.
Logical errors in buildapp.xml	• Trying to generate a runtime that is not allowed for the application by the CDKey • Creating a runtime installation before the application was ever compiled	• Manually execute the same operations from the ACE UI.

We recommend performing the operations the first time manually, using ACE, in the same order as in the buildapp.xml, and verifying that they work properly, before using the automatic mode.

Limitations

The following known limitations exist:

- The feature is certified only for the following product flavors:
  - JIS XHTML for 5250
  - JIS Java for 5250
  - JIS XHTML for 3270
  - JIS Java for 3270
- The main window of ACE is still visible in automatic mode.
- The operation *Create J2EE Runtime Installation* requires that you Generate Runtime for the application and all of its libraries at least once from within ACE. Otherwise, you get the following Perl error:

  ```
  "Error: Key 'libraries' not found in section 'program' at …\\perl\gen/HierarchyFile.pl line 69".
  ```

  Workaround:

  Generate the runtime in automatic mode, then edit the runtime INI (<application>.ini) and add the following setting:

  ```
  [Program]
  Libraries=<semicolon separated list of libraries>;
  ```

If you do not have any libraries, then the list should just contain the name of the application. For example, Libraries=TEST; for an application named TEST.

- Do not use comments (<!-This is a comment->) in the buildapp.xml file.

- When the packaging of the J2EE runtime installation fails, remote.log still reports successful completion.
- When using the command line interface, if you set the NewAndModified attribute to 1 the first time you generate runtime, the compilation fails. The first time you compile you must not use NewAndModified.

Improved User Interface

Introduction

The JIS Java Client user interface has been improved to create a more modern look and feel for the JIS Java components and to improve the table components functionality.

Note: Excluding the mouse wheel support explained below, all of the other features are disabled by default in order to maintain backward compatibility.

To enable the new JIS look & feel add the following Applet parameter:

```
<PARAM name = "ThemeName" value = "default">
```

In addition JIS now offers the ability to customize the look & feel per project specific requirements. To achieve this, users will need to implement the UIManager interface or extend the DefaultTheme class.

Setting the following Applet parameters activates a user defined Theme class:

```
<PARAM name = "ThemeName" value = "UserDefined">
<PARAM name = "UserDefinedThemeClassName" value = "Fully qualified
name of the Theme class">
```

To learn more about implementing project specific look & feel we recommend:

1. Reading the JavaDoc for interface cst.gwt.general.UIManager and for class cst.gwt.general.DefaultTheme in the JIS client Java Doc (located in ..\JacadaFiles\Docs\Client).

2. Reviewing the sample theme classes provided as part of the JIS samples (located in ..\JacadaFiles\samples\features\themes).

These user interface improvements include:

Mouse rollover mode

Controls such as arrow buttons, scrollbars, checkboxes and radio buttons will now change their look when the mouse rolls over them. This feature is only enabled when a Theme name is defined.

Mouse wheel support

Mouse wheel support is incorporated into window scrollbars and table scrollbars when using cached tables. This feature is only enabled when using JDK 1.4 and above.

Arrow buttons and scroll bars

The arrow buttons used in prompts, combo boxes, date fields and scroll bars are now displayed using a more modern look & feel.

Rounded rectangles for group boxes and frames

Group box, frames and tab headers now support “rounded corners”. This feature is only enabled when a Theme name is defined.

Calendar control

The calendar control’s look has been improved.

Client persistent storage

Changes to the table component including column resizing and reordering can now be saved locally on the user’s machine. Changes made by the user during the session lifetime will be saved to the disk when a session ends, and reloaded when a session starts. In addition, the user may save the current state using the **File**->**Save Storage** menu item and restore the default state using the **File**->**Clear Storage** menu items.

To enable the persistent storage feature add the following Applet parameter:

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

The persistent storage feature also requires working with the signed Applet and JDK 1.4 and above.

The persistent storage information is saved in a file named JacadaClientProperties.xml created in the user’s temporary files directory.

Table column reordering

This feature allows interactive table column reordering during runtime by using the mouse to drag and drop columns. In order to enable table columns reorder during runtime:

1. Make sure that the Drag columns checkbox is checked in the table control style tab in ACE.

2. Add the following Applet parameter:
<PARAM name = "AllowTableColumnDragging" value = "true">

The column order initially defaults to the column order specified in ACE.

When the client persistent storage feature is enabled, the new column order is saved for the specific client machine.

Note the following limitations:

- Folded table columns cannot be reordered.
- When using fixed columns, the fixed columns cannot be reordered.

Table row sorting

When using fully cached tables, it's useful to sort the table data by column. This is now possible by clicking the column header of the column to be sorted. Clicking the column header once, sorts the table by ascending order, clicking again sorts the table by descending order. In order to return to the original host row order you must reload the table.

This feature is of limited use for most host tables since on normal tables it only sorts the current table page.

To activate this feature add the following Applet parameter:

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

Sorting order is not maintained after exiting the current screen.

Known Limitations

- Heavy weight controls such as non transparent frames do not comply with the new look.
- Multiline labels are based on the java.awt TextArea component which draws its own scroll bars and therefore cannot be adapted to the JIS look and feel.

Additional Enhancements

Accessing the application server's HTTP session from an XHTML extension (ATL-27368)

It is now possible to access the application server's HTTP session from XHTML extensions:

```
/**
* Get J2EE sessionId
* @return sessionid
*/
 public String getSessionId ();
/**
* Binds an object to this session, using the name specified.
*/
public void setSessionAttribute(String name, Object value);
/**
* Returns the object bound with the specified name in this session,
* or null if no object is bound under the name.
*/
public Object getSessionAttribute(String name);
```

Loading resources from an XHTML extension (ATL-27368)

It is now possible to load resources from an XHTML extension:

```
/**
* This method allows to read resource using ApplicationResourceLoader
* could be used in prop and J2EE server.
* @return InputStream
*/
public InputStream getResourceAsStream(String resource);
```

For example:

```
package appls.DEMO1.xhtml.user;


import com.jacada.jis.runtime.server.frontend.xhtml.context.*;

import com.jacada.jis.runtime.server.frontend.xhtml.general.DocumentBuilderProvider;

import java.io.InputStream;

import org.w3c.dom.Document;

import org.w3c.dom.NodeList;

import cst.debug.Debug;


public class Appl implements
com.jacada.jis.runtime.server.frontend.xhtml.extension.IUserPageExtension {


   public Appl () {}


   public void onPageLoad(OnPageLoadContext context) {

      Document staticXml = null;

      String xmlFilePath = "appls/"+ context.getLibraryName() +"/xml/"
+context.getSubApplName()+ ".xml";

      InputStream is = context.getResourceAsStream(xmlFilePath);

      try {

        staticXml = DocumentBuilderProvider.getDocumentBuilder().parse(is);

      } catch(Exception e) {}

      context.setSessionAttribute("staticXml", staticXml);

      // do some processing ...

      Document staticXmlFromSession =
(Document)context.getSessionAttribute("staticXml");

      NodeList nl = staticXmlFromSession.getChildNodes();

      Debug.print(1, "staticXml from HttpSession " + nl.item(0));

   }


   public void onPageSubmit(OnPageSubmitContext onSubmitContext) {}


}
```

Known Limitations

The new APIs cannot be used from onPageSubmit().

XHTML Date control enhancements (EU-05423)

1. The calendar window default style:

The default style for the calendar window can be overridden from an external CSS file (calendar.css). The calendar.css is created during runtime generation process, in the appls/<APPLNAME>/xhtml/CSS. By default, the file consists of only one line comment, to allow browsers to cache it.

For example, if you want to increase the font size of the day numbers in the calendar window you can do it by defining a CSS definition:

```
.day font{
    font-size : 150%;
}
```

Following is a list of CSS class names in the calendar window:

currentDay – class name for current day

weekendDay – class name for weekend days

day – class name for all other days

weekday – class name for the titles of the week days(Sun, Mon)

currentDate – class name for the current date title (August 2007)

2. It is now possible to use the “.” character as a date separator.

3. The default date format is selected according to the browser locale.

Displaying digits in Java Client spin box controls (ATL-28026)

When you require adding a leading ‘0’ in front of a single digit number in a spin box control, perform the following:

```
appls.<APPLNAME>.user.ApplSubApplwindow.java
public void setControl(Component comp, int tabIndex) {
         super.setControl(comp, tabIndex);
         if (comp instanceof GUISpin) {
             GUISpin spin = (GUISpin)comp;
             spin.setInputRestrictor(new NumberInputRestrictor("0#;"));
         }
}
```

Support for monochrome terminals (ATL-28319)

Added support for monochrome terminals. Set the following in the runtime-ini file to configure a model 5 monochrome terminal:

```
[GUISys TN5250]
TerminalType=IBM-3477-FG
```

SSL Connection (ATL-29082)

It is now possible to enable the SSL connection between the server and the host by simply configuring the cst.server.com.CSTSSLSocketFactory class to be the socket factory implementation. For example:

```
[GUISys TN5250]
SocketImplFactory= cst.server.com.CSTSSLSocketFactory.
```

Refer to the Appendix for further details.

SSL connection limitations:

The built-in CSTSSLSocketFactory solution does not support:

1. Client side of SSL encryption.

2. ProxyServlet to server encryption.

3. Server side of client to server SSL.

It's only designed to solve the secured Telnet use case. All other combinations will still need to use extensions of the socket factory.

Changing XHTML Message Boxes (ATL-27839)

A new feature has been added which enables manipulating the text of a message box or completely eliminating the message box using an XHTML extension.
For Example:

```
file src\appls\<app-name>\xhtml\user\Appl.java:


public void onPageLoad(OnPageLoadContext context) {
   Window window = context.getWindow();
   XhtmlControl control = window.getControlByName("messageBoxText");
   if (control != null) {
      if (control.getText().equalsIgnoreCase("Test message box")) {
         control.setText("");
      }
   }
}
```

This extension will remove the message box if it contains the text "Test message box" by changing its text to "".

Menus in XHTML (ATL-28670)

Infrastructure has been added to create a menu bar using in an XHTML extension.

New Features in Service Pack 9.0A07

Refreshing the XHTML Client When a Page on the Host is Updated

The client-oriented architecture of HTML means that requests always originate from the client. A state change on the host such that the host updates the server with a new screen was not displayed on the client unless the client sent a request to the host.

Session changes on the host are now “pushed” to the XHTML client, without the client requesting the updated server page. A new connection is opened to the server and if there is a screen change on the host, the new page is sent to the client immediately.

To enable pushing pages from the host to the client, in the [XHTML] section of the RUNTIME.INI file set the EnableAutomaticServerUpdates parameter to 1.

Note: Enabling this feature may impact significantly on scalability since it requires the client to maintain an open request (an open thread) to the server for the entire session.

New Features in Service Pack 9.0A06

Support for Keyboard Buffering

Many host emulators enable keyboard keystrokes to be buffered, so that users can continue typing without waiting for the host screen to refresh. After the host screen is refreshed, the content of the buffer is played back as if it was typed at that moment.

The Java client, keyboard buffering feature, requires an updated runtime license. Contact Software AG to obtain the runtime license file. Place the license.dat file in the <JISRoot>\JacadaFiles\classes folder.

Activating Keyboard Buffering

After installing the new license key that enables keyboard buffering, you must change parameters in the html file that launches the application, to activate the feature.

The following parameters influence how keyboard buffering functions.

Table 1-1: Parameters for keyboard buffering

Parameter	**Default**	**Value Description**
`UseEventDispatchThread`	true	You must set this parameter to true to use keyboard buffering. When set to true, all server requests are dispatched on the event AWT dispatch thread.
`EnableKeyboardBuffering`	true	This parameter must be set to true to use keyboard buffering. Set to false to turn off keyboard buffering.
`KeyboardBufferingResetKey`	Escape	The reset key code. Pressing the specified key resets the playback buffer. Valid codes can be obtained using: `KeyEvent.getKeyText(<Virtual key code>)`
`KeyboardBufferingResetKeyModifier`	Shift	Reset the key modifier. Valid codes can be obtained using: `KeyEvent.getKeyModifiersText(<Virtual modifier>)`
`HideKeyboardBufferingToolbar`	true	Set this parameter to false to show the keyboard buffering toolbar when testing the application.

Known Limitations

The following limitations influence how keyboard buffering functions:

• The product's keyboard buffering feature must replace existing extensions that provide keyboard buffering functionality.

• Replace every occurrence of JacadaStarter activate(..., true) API in the client extensions to JacadaStarteractivate(..., false).

• Manipulating key events or focus events using Java extensions can adversely impact keyboard buffering.

• Buffering key events starts after the display of the first window.

New Features in Service Pack 9.0A05

API available to trigger server methods

An API is now available to enable triggering a given method on the server directly from the Java client, without the need to trigger a menu item, button or accelerator linked to the method:

```
/**
* Activate method on the server by trigger id.
* The trigger id can be obtained as follows:
* For subapplication specific methods: from the
* <subapplication>.sa file
* For GUTMs: from applicat.ion
* @param id trigger id of the method as written in the
* applicat.ion or .sa file.
* @param wait true to lock the current thread's execution until
* the action is finished.
* @exception IllegalStateException if the method is invoked from
* the CommServer thread.
*/
public void activate(int id, boolean wait)
```

Printing

A number of changes were made to the way printing is handled.

Support for portrait and landscape printing

The Printer setup dialog for the Java and XHTML clients now supports printing both portrait and landscape printouts.

Enabling specifying margins

You can now specify the margins in the Printer setup dialog instead of using the default one inch margin.

Enabling specifying the paper size

You can specify the paper size in the Printer setup dialog or in the applet parameters, using the PrinterEmulationPaperType setting:

```
<PARAM name = "PrinterEmulationPaperType" value = "A4">
<PARAM name = "PrinterEmulationPaperType" value = "LETTER">
```

Improved handling of underlines in print output

Underlining in a print output now works. For example, characters are no longer misaligned.

Enabling printing to any printer

You can now print to any printer defined on the network and not just to the default printer.

New printer parameters

The following parameters have been added to the application INI file:

• DynamicCalculateWidth

When set to 1, the server calculates the width of each page in the print job, resulting in a better layout of each page. Note that this may cause a non-homogeneous look to the whole job, since different pages may have different font sizes (due to a difference in the actual text width).

• IgnoreEMAtStartOfLine in the [TN3270 section]

When set to 1, the printer emulation suppresses new line and carriage return characters, when they appear in the data stream, following the Mainframe “End of Medium” signal.

New Features in Service Pack 9.0A02

Enabling reconnecting to a database after the connection or session fails

When using external data methods to connect to a database, by default all sessions in a given server process, reuse the same JDBC connection. If this connection fails, every session using this connection may not be able to connect to the database.

To recover from a connection failure, the following new methods are available:

RepairDBConnection and RepairDBSession. The following example shows the usage of these methods:

```
Action: Query_with_close
Trigger: 17001 WaitIndicator: True
ScrambleName: False MoveMode: MoveNone
Description: ‘’
```

```
#0 = DoMethod: Receiver: ‘EXTERNALDATA’ Method: AllocDBSession Parms: (
‘"jdbc:as400://10.90.17.18;libraries=LY,*LIBL"’ , ‘"USER"’ , ‘"PSSWRD"’
)
If: Cond: ‘#0 == _FAIL ’
#0 = DoMethod: Receiver: ‘ExternalData’ Method: RepairDBConnection Parms:
( ‘"jdbc:as400://10.90.17.18;libraries=LY,*LIBL"’ , ‘"USER"’ ,
‘"PSSWRD"’ )
EndIf:
#3 = DoMethod: Receiver: ‘#0’ Method: ExecuteQuery Parms: ( ‘"select *
from sections"’ )
If: Cond: ‘#3 == -1’
#0 = DoMethod: Receiver: ‘ExternalData’ Method: RepairDBSession Parms: (
‘#0’ )
#3 = DoMethod: Receiver: ‘#0’ Method: ExecuteQuery Parms: ( ‘"select *
from sections"’ )
EndIf:
Do: Times: ‘1000’
#4 = DoMethod: Receiver: ‘#0’ Method: Next Parms: ( )
If: Cond: ‘#4 == _FALSE ’
Break:
Else:
#5 = DoMethod: Receiver: ‘#0’ Method: GetStringByColumnIndex Parms: ( ‘1’
)
DoMethod: Receiver: ‘System’ Method: DebugPrint Parms: ( ‘1’ , ‘"row:" +
#5’ )
EndIf:
EndDo:
DoMethod: Receiver: ‘#0’ Method: Close Parms: ( )
```

Limiting the size of the server log files

The server can now create a new log file each time the size of the current log file exceeds a predetermined limit. The current log is renamed using a revision number.

The maximum size for the current log before a new log is created is set via the setting, RtDebugFileMaxSize. When RtDebugFileMaxSize=0, logging is always performed to a single log file, unlimited in size.

There is also a setting, for the maximum number of files that a server process is allowed to create. The RtDebugMaxFiles setting specifies the number of log files that the server can create.

These parameters can be set in the jacadasv.ini file or from the command line, as follows:

Via jacadasv.ini – Specify the parameters in the [GeneralParameters] section:

```
[GeneralParameters]
RtDebugFileMaxSize=<max_size_of_log_file_in_bytes>
RtDebugMaxFiles=<number of log files>
```

Via the standalone server command line – Specify the following switches for the parameters:

```
RtDebugFileMaxSize –m<max_size_of_log_file_in_bytes>
```

```
RtDebugMaxFiles -b<number of log files>
```

New Features in Service Pack 9.0A01

Enable opening a window in a maximized state

The Java Client now enables writing a Java extension for opening the Applet window in maximized state.

To maximize a window, use the following code:

```
package appls.TEST.user;

import java.awt.*;
import cst.gwt.*;

public class MainWindow extends appls.TEST.original.MainWindow{
      boolean firstShow = true;

      public void setVisible (boolean show) {
            if (show && firstShow) {
                  firstShow = false;
                  ((GUICSTFrame)getMyFrame()).setExtendedState(Frame.MAXIMIZED_BOTH);
            }
            super.setVisible(show);
      }
}
```

Register the MainWindow.java extension in the JacadaStarter using the addWindow() API

Deploying a service to a J2EE Server

The HTML page generated by JIS contains links to resources such as images, script files, css files, Html files, etc. The URLs specified by these links differ between the standalone server and J2EE deployment. In the standalone server the URLs start with ‘classes’ (for example, src="/classes/js/jacada.js", href="/classes/appls/NRT/xhtml/CSS/kb_IE.css") while in J2EE deployment the URLs start with the name of the application (for example, src="/NRT/js/jacada.js", href="/NRT/appls/NRT/xhtml/CSS/kb_IE.css").

All URLs created by the server code are automatically adjusted, based on the server type (standalone or J2EE). However, URLs added through user extensions (java extensions, javascripts, htmls, etc.) should be changed as follows to ensure that the same code base will function properly for both the standalone server and a J2EE server.

• In XHTML java extensions:

Use an API method that returns the correct URL prefix: for the standalone server it returns /classes and for the J2EE server it returns /<APPLNAME>.

```
/**
* Get the application root dir
* @return String /classes - for standalone server, /<APPLNAME>
for J2EE Server
*/
public String getApplicationRootDir ();
```

• In javascript extensions:

Use javascript similar to the following:

```
window.addExternalJavaScriptFile(context.getApplicationRootDir() +
"/appls/TABLE/resources/test.js");
```

• In the XHTML html extensions:

Use the template word $ApplicationRootDir, which is changed to "/

classes" or to "/<APPLNAME>", as follows:

```
<script language="JavaScript1.2" type="text/javascript"
src="$ApplicationRootDir/appls/TABLE/resources/test1.js"></script>
```

New Features in Service Pack 9.0A00

XINIT keyword in BMS maps now supported

The screen image creation process now supports the presence of the XINIT keyword in a BMS file, without codepage considerations. The XINIT keyword in BMS maps allows the mapfield to be initialized to a hexadecimal value. A character with a value less than 0x40 is considered to be a terminal control character and is replaced with a blank.

Maximum permitted size of ACE method increased

The maximum number of lines in a single Ace method has been increased to 2048. Until now, the limit was 512 lines.

Print setup dialog can be skipped

When printing the current window, the Java client always shows the Printer Setup dialog, allowing the user to choose various printing options, including page type, page orientation, print destination, and so on.

You can now eliminate the display of the printer setup dialog if you so choose. To eliminate the display of the printer setup dialog, add the following parameter to the html page:

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

The default value is "true".

New methods for setting colors of selected cells

New APIs have been added to the Java client GUITable component for setting the background and foreground colors of selected cells.

To set the background color in which selected cells are to be painted:

```
public void setSelectionBackground(Color color)
```

To set the foreground color in which selected cells are to be painted:

```
public void setSelectionForeground(Color color)
```

To discover if the default background color for selected cells has been changed (by setSelectionBackground or any other method):

```
public boolean isDefaultSelectionBackground()
```

To discover if the default foreground color for selected cells has been changed

(by setSelectionForeground or any other method):

```
public boolean isDefaultSelectionForeground()
```

To discover the current background color of selected cells:

```
public Color getSelectionBackground()
```

To discover the current foreground color of selected cells:

```
public Color getSelectionForeground()
```

The following APIs are now marked deprecated since they only control the selection background color but not also the foreground color:

public void setSelectionColor(Color color) has been replaced by setSelectionBackground (color color)

public boolean isDefaultSelectionColor() has been replaced by public boolean isDefaultSelectionBackground()

public Color getUserSelectionColor() has been replaced by public Color getSelectionBackground()

A good place to use these APIs is by overriding the setContol() or createGUIControls() methods.

Example:

```
public void setControl(Component comp, int tabIndex) {
      super.setControl(comp, tabIndex);
      if (comp instanceof GUITable) {
             GUITable table = (GUITable)comp;
             table.setSelectionBackground(Color.blue);
      }
}
```