



J A D E TM

Migrating to JADE Consolidated Release Information

VERSION 6.3.06



Copyright © 2010
Jade Software Corporation Limited
All rights reserved

Jade Software Corporation Limited cannot accept any financial or other responsibilities that may be the result of your use of this information or software material, including direct, indirect, special or consequential damages, or loss of profits. There are no warranties extended or granted by this document or software material.

You should be very careful to ensure that the use of this software material and/or information complies with the laws, rules, and regulations of the jurisdictions with respect to which it is used. No part of this document may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without the express written permission of Jade Software Corporation Limited.

The information contained herein is subject to change without notice. Revisions may be issued to advise of such changes and/or additions.

Copyright © 2010 Jade Software Corporation Limited.
All rights reserved.

JADE is a trademark of Jade Software Corporation Limited. All trade names referenced are the service mark, trademark, or registered trademark of the respective manufacturer.

For details about other licensing agreements for third party products, you must read the JADE **ReadMe.txt** file.

Contents

JADE Release Support	9
Deimplementations and Deprecations	9
Exposure Deimplementation.....	9
JADE Portable Graphical User Interface (GUI) Client	9
PKWare Compression.....	9
Real[10] Parameters in External Function Calls.....	10
RPS SQL Script Execution	10
Accessing Details about Faults Fixed in Releases	11
How to Locate PARs Fixed in a Specific Release.....	11
Upgrading to JADE 6.3.06 from a JADE 6.2 or Earlier JADE 6.3 Release	12
Upgrading to JADE 6.3 from a Windows JADE 6.2 or Earlier JADE 6.3 Release.....	12
Running Two Windows Releases of JADE on the Same Workstation.....	15
Upgrading to JADE 6.3 from a Linux JADE 6.2 or Earlier JADE 6.3 Release	15
JADE Thin Client Upgrade	16
Upgrading a 32-Bit Presentation Client Connecting to a 64-Bit Application Server	16
Upgrading a Synchronized Database Environment (SDE).....	17
Upgrading an RPS Node from JADE 6.2 to 6.3	17
Upgrade Validation.....	18
Hot Fix Releases	18
Changes in JADE Release 6.3.06	19
Highlights in Release 6.3.06.....	19
Converting Pictures	19
Debugging an Application.....	19
Development Environment Changes	19
Adding a Method from a Free-Standing Method Editor Pane	20
Adding a Schema	20
Class Browser	20
Closing MDI Child Forms from the Windows Menu	20
Creating Patch Files when Patch Versioning is Enabled	21
External Function and Library Extraction	21
Finding a Global Constant	21
Listing Only the Most-Recently Accessed Methods in the History List	21
Patches Browser.....	21
Text Templates	22
Error Message.....	22
14217 - This feature is not available in this environment.....	23
Exception Handling	23
Executing Conditional Code.....	23

Changes in JADE Release 6.3.06, continued

File Handling	23
Browsing for Files	23
Maximum Size of Data on a Presentation Client.....	24
Forms and Controls	24
ComboBox Maximum Length	24
Control Skins	24
JPEG2000 Picture Format Support.....	24
Interfaces	24
Prefixing Stub Methods	24
Specifying a Text Template for Interfaces.....	25
JADE Dynamic Objects.....	25
JADE Initialization File.....	25
[FaultHandling] Section.....	25
JADE Interpreter Output Viewer.....	26
Application:: <i>clearWriteWindow</i> Method.....	26
Application:: <i>closeWriteWindow</i> Method.....	26
Database Path Display	26
Restricting Font Selection.....	26
Saving the JADE Interpreter Output Viewer Contents.....	27
JadeBytes:: <i>getStatistics</i> Method.....	27
JadeHTMLClass:: <i>generateHTMLForJadeTagsOnly</i> Property.....	27
Locking.....	30
Relational Views.....	30
SDS and Recovery Considerations.....	30
RPS Considerations	32
Stripping Source Code during a Batch Load	32
Terminating an Application.....	33
Thin Client Automatic Downloads.....	33
Web Services	33
Generating a Web Service Consumer Unit Test Class and Stub Methods	33
JadeWebServiceProvider:: <i>getLastStatistics</i> Method	38
Support for the Document/Literal Bare SOAP Format.....	40
Support for the RPC/Literal SOAP Format	41
C-Level Application Programming Interface (API) Enhancement	41
Getting the Name of the Initialization File	41
Getting a Boolean Value from the Initialization File.....	42
Getting a Signed Integer Value from the Initialization File.....	42
Getting an Unsigned Integer Value from the Initialization File	42
Getting a String Value from the Initialization File	43
Setting a Boolean Value in the Initialization File.....	43
Setting a Signed Integer Value in the Initialization File.....	44
Setting an Unsigned Integer Value in the Initialization File.....	44
Setting a String Value in the Initialization File	45
Getting the JADE HOME Directory.....	45
Getting the JADE Installation Directory.....	45
Getting the JADE Lib Directory	46
Getting the JADE Temp Directory	46
Converting a JADE Character to an ANSI Value.....	46
Converting a JADE Character to a Unicode Value	47

Changes in JADE Release 6.3.05	48
Highlights in Release 6.3.05	48
.NET Component Importing	48
JadeDotNetType:: <i>createBoxedPrimitive</i> Method	48
JadeDotNetType:: <i>getBoxedPrimitive</i> Method	49
Non-Default Constructors on Imported .NET Components	49
Batch JADE Database Utility	49
listBackupinfoFileNames Command	50
makeBackupinfo Command	50
New Error Messages	50
Code Coverage	51
Compact JADE	52
Control Changes	52
Combo Box List Entries	52
Disabled Text Color of Controls under Portable GUI	53
Drawing on Picture Controls	53
JadeEditMask and TextBox Classes	54
JadeRichText:: <i>linkClicked</i> Method	54
Looking Up a Control on a Form by Name	54
Delta Databases	55
Getting Started	55
JADE Initialization File Parameters	56
[JadeServer] Section	56
DeltaDatabaseCapable Parameter	56
ResetDeltaModeOnRestart Parameter	56
[DeltaDb] Section	56
Activating a Delta Database	56
Deactivating a Delta Database	57
SDS and RPS Operational Characteristics	58
Database Tracking and RPS Replication	58
Administrative Restrictions	58
New SDS_ReasonDeltaModeEntered Tracking Stopped Reason Code	58
Caveats that Apply when Using Class Extent Methods in Delta Mode	59
New Methods Defined in the System Class	59
<i>activateDeltaDatabase</i> Method	59
<i>getDeltaDatabaseStatus</i> Method	60
Restarting a Delta Database	60
JADE Database Batch Utility (jdbutilb) Command	61
<i>clearDeltaMode</i> Command	61
JADE Database Administration Utility (jdbadmin) Commands	61
<i>ActivateDeltaDb</i> Command	61
<i>DeactivateDeltaDb</i> Command	61
Development Environment Change	62
Splitting the Editor Pane View	62
Encoding and Decoding the Base64 Format	62
Binary:: <i>base64Encode</i> Method	62
Binary:: <i>base64EncodeNoCrLf</i> Method	63
String:: <i>base64Decode</i> Method	63
Base64 Encoding and Decoding Example	63
Error Messages	64
1040 - Failed to open schema agent	64

Changes in JADE Release 6.3.05, continued

1160 - Background Process request was not completed	64
1161 - Background Process request encountered a lock error	64
1265 - Environmental object operation is out of scope for process	64
3219 - Cannot run SDS secondary server in SingleUser	65
6429 - Cannot mark the default map as partitionable	65
Fault Handling	65
TCP/IP Disconnect Logging Suppression	65
Thread Execution and Callback Logic Exception Handling	65
HTML Documents	65
Finding and Updating HTML Document Source	66
Translating Strings in the HTML Document Source	66
JadeAuditAccess Class	66
JadeBytes Class Changes	67
JadeHttp Configuration	67
Merge Iterator Restrictions Removal	67
ODBC Changes	68
ODBC Driver	68
Thin Client Data Source Name (DSN)	68
ODBC Execution Trace Output	68
Process:: <i>isRunningScript</i> Method	69
Relational Population Service (RPS)	69
Automatically Restarting the Datapump Application	69
Fault Handling when the Datapump Application Terminates Abnormally	70
Running a User Data Pump Extract on an SDS Secondary or RPS Node	70
Report Writer	71
Schema:: <i>extractControlIdsCSV</i> and <i>extractControlIdsCSVforSchema</i> Methods	71
Synchronized Database Service (SDS) Tracking	72
System Sequence Numbers	72
System:: <i>getSystemSequenceNumberNext</i> Method	73
System:: <i>createSystemSequenceNumber</i> Method	74
System Sequence Number Errors	74
1454 – The SystemSequenceNumber name is invalid	74
1455 – The SystemSequenceNumber initial value cannot be negative	74
1456 – The SystemSequenceNumber has reached the maximum value (Max_Integer64)	74
Update Locks	74
Updating a Partitioned Database File	75
Web Services	75
Creating and Deleting Virtual Directory Files	75
<i>createVirtualDirectoryFile</i> Method	75
<i>isVDFFilePresent</i> Method	76
<i>deleteVirtualDirectoryFile</i> Method	76
Handling Web Service Consumer UTF-8 Conversion Errors	76
Maximizing the Web Service Exposure Wizard	77
Removing Web Service Consumers	77

Changes in JADE Release 6.3.04**78**

Highlights in Release 6.3.04	78
Application Class <i>doWindowEvents</i> Method	78

Changes in JADE Release 6.3.04, continued

Classes In Use Browser Performance.....	79
Defining a Web Service Exposure.....	79
Disabled Text Color of a Text Box under Portable GUI.....	79
Error Messages.....	79
3115 - GetObject request buffer too small.....	79
6428 - Cannot access versioned feature in current version of method.....	79
Excluding Offline Objects when Iterating a Collection.....	80
JADE Initialization File.....	80
Object: <i>hasMembers</i> Method.....	80
Partition Awareness.....	80
Patch Number Batch Extract.....	80
Profiler Cache Statistics.....	81
Relational Population Service (RPS).....	81
Process: <i>isUserDataPump</i> Method.....	81
Relational View.....	81
Controlling the Size of a Result Set Returned by the ODBC Driver.....	81
Property Details Access.....	81
Renaming or Editing User-Defined Table Names.....	82
Release Notes Splash Screen Display.....	82
Reorganization Updates.....	82
Thin Client Download Process.....	83
Web Services.....	83
Word Wrapping in Tables.....	83
WSDL Nested complexType Definition Load.....	83
ZLIB Compression on Compact JADE.....	84

Migrating to JADE Release 6.3.06

This document covers the following topics.

- [JADE Release Support](#)
- [Accessing Details about Faults Fixed in Releases](#)
- [Upgrading to JADE 6.3.06 from a JADE 6.2 or Earlier JADE 6.3 Release](#)
 - [Upgrading to JADE 6.3 from a Windows JADE 6.2 or Earlier JADE 6.3 Release](#)
 - [Upgrading to JADE 6.3 from a Linux JADE 6.2 or Earlier JADE 6.3 Release](#)
 - [JADE Thin Client Upgrade](#)
 - [Upgrading a Synchronized Database Environment \(SDE\)](#)
 - [Upgrading an RPS Node from JADE 6.2 to 6.3](#)
 - [Upgrade Validation](#)
 - [Hot Fix Releases](#)
- [Changes in JADE Release 6.3.06](#)
- [Changes in JADE Release 6.3.05](#)
- [Changes in JADE Release 6.3.04](#)

Refer to [RelInfo6303.pdf](#) in your JADE **documentation** directory for details about JADE release 6.3.03 changes that may affect your JADE 6.2 existing schemas and changes in JADE release 6.3.03.

Tip For information about using Acrobat Reader to view JADE documents, see “[JADE Product Information Library in Portable Document Format](#)”, in Chapter 2 of your JADE *User's Guide* in your JADE **documentation** directory.

The JADE *Product Information Library* document ([JADE.pdf](#)) provides a summary of JADE product information library documents and navigation to them.

If you want to develop your own installation process:

- For Windows, the JADE install and upgrade steps are documented in the **ReadmeInstallSteps.txt** file in the **\documentation** directory.
- For Linux, the steps are as executed in the **pre_i** and **post_i** scripts in the **bin** directory in the **/opt/jade** subdirectory for the release.

To customize the deployment upgrade on Windows, see Appendix A, “[Customizing the Deployment Upgrade Process](#)”, in your JADE *Runtime Application Guide*.

JADE Release Support

With the release of JADE 6.3, support for JADE 6.1.15 (the final JADE 6.1 release) continues until October 2010. Prior versions of JADE 6.1 will not be supported.

For details about the JADE release policy, go to:

http://www.jade.co.nz/downloads/jade/JADE_ReleasePolicy.pdf

For details about the JADE release schedule, go to:

<http://www.jade.co.nz/jade/updates.htm#releasesched>

Microsoft support for Windows 2000 ends on July 13, 2010. If you are still using this operating system, you should migrate to a later operating system version before that date as JADE will not be supported on Windows 2000 when Microsoft support for Windows 2000 ends.

Deimplementations and Deprecations

This section contains the deimplementations and deprecations in JADE 6.3.

Exposure Deimplementation

Notice is given of the intent to deimplement JADE's ActiveX exposure feature, starting from JADE release 7.0. This means that from JADE 7.0, the ActiveX exposure feature will become unavailable.

The ActiveX exposure enables you to expose selected features of your JADE system to application development tools such as Microsoft's Visual Basic and C++ languages through ActiveX technologies. JADE implements ActiveX generation through the wizard feature of Microsoft's Visual Studio 6. However, this product is no longer supported by Microsoft. In addition, new users of JADE may be unable to source a copy of Visual Studio 6.

In recent years, ActiveX technologies have been replaced by .NET. From JADE 6.3, you can now generate exposures using these .NET technologies. This provides a more modern, flexible, and easier to develop mechanism than that provided by ActiveX.

It is recommended that, where required, you re-write ActiveX exposures using the new JADE .NET exposure. For details, see [Chapter 19](#) of the *JADE Development Environment User's Guide*.

JADE Portable Graphical User Interface (GUI) Client

JADE portable GUI client is deprecated for both Linux and Windows platforms.

In the second half of 2009, we plan to release a Silverlight-based thin client that will enable JADE user interfaces to be deployed in Web browsers via Silverlight and Moonlight. Silverlight is Microsoft's cross-browser, cross-platform plug-in for delivering media and rich user interfaces to the Web.

The Silverlight plug-in is freely available for all major browsers including Internet Explorer (IE), Firefox, and Safari on Windows and the Macintosh operating system; and the Mono Project has released Moonlight, which is an open source implementation of the Silverlight plug-in for Firefox on Linux.

PKWare Compression

As notified in JADE 6.2 release information, PKWare compression is not supported in this release.

Before upgrading to JADE 6.3:

1. Change existing code in your JADE 6.2 applications to use the **compressToBinary** method of the **Binary**, **String**, and **StringUtf8** primitive types and the **uncompressToBinary**, **uncompressToString**, and **uncompressToStringUtf8** methods of the **Binary** primitive type.
2. If you have persistent data that has been compressed using the PKWare compression libraries, update that data by uncompressing it using the appropriate **uncompress**, **uncompressString**, or **uncompressStringUtf8** decompression method, and then recompress it using the **compressToBinary** method using a compression option from the **Binary** primitive **Compression_Zlib**, **Compression_ZLibFast**, or **Compression_ZLibSmall** constant.

Note JADE strongly recommends that you make the changes necessary to transition to the use of zlib compression.

However, if you elect to continue to use PKWare compression, you should be aware that before you can upgrade to a 64-bit version of JADE, coding changes to use new methods and data recompression are necessary. For details about using PKWare compression in 32-bit editions of JADE 6.3, see “Compression and Decompression Methods Deimplemented”, in the JADE 6.3.03 Release Information document (that is, [RelInfo6303.pdf](#)).

Real[10] Parameters in External Function Calls

As notified in JADE 6.2 release information, external function calls with **Real[10]** parameters are no longer supported.

The upgrade validation process checks external functions for **Real[10]** parameters, logs any detected usages, and the upgrade fails. You must change the usages to **Real[8]** and re-run the validation.

RPS SQL Script Execution

In this release, **sqlcmd** has replaced the Open Database Connectivity (ODBC) interface as the default mechanism for SQL script execution on an RPS node.

SQL scripts are used to create or alter table definitions and to load data. These scripts can be executed from the RPS Manager utility or automatically by the **Datapump** application.

The SQL Server **sqlcmd** utility is the preferred mechanism for SQL script execution. In order to use **sqlcmd**, it must be installed on the machine hosting the RPS node. The SQL Server instance name is specified from the RPS manager node configuration dialog.

The advantages of using **sqlcmd** are as follows.

- Error results, which are lost when using the ODBC interface, are correctly reported.
- The error information from SQL Server is saved in a log file.

Use of the ODBC interface for script execution is supported in JADE release 6.3. However, it will be deimplemented in JADE 7.0.

Accessing Details about Faults Fixed in Releases

To access the complete documentation about the PARs fixed in this release, you can directly access **Parsys**, our Fault Managements and Customer Contact system. This enables you not only to view the progress of your own contacts but also to view all of the PARs fixed in a specific release.

If you have any queries about **Parsys**, please direct them to JADE Support in the first instance.

You can download the install shield for **Parsys** from the following URL.

http://www.jade.co.nz/Jade6/jade6_parsys.htm

When you first run the **Parsys** application, it downloads an update via the automatic thin client download feature. When this has completed and you have the log-on form ready and waiting, please contact JADE Support, who will then send you an e-mail message with your user code and password details. **Parsys** requires you to change your password when you first log on.

Note Because the encryption of passwords is a one-way algorithm, we cannot advise you of your password should you forget it, but we can reset it to a known value again.

How to Locate PARs Fixed in a Specific Release

This section describes the actions that enable you to locate Product Anomaly Reports (PARs) fixed in a specific release.

➤ To perform an advanced search

1. Select the **Advanced Search** command from the Search menu with the following settings on the **Basic Search Criteria** sheet.
 - a. The **Latest** option button is selected in the Mode group box.
 - b. **All** is selected in the **Priority** list box.
 - c. The **PAR** check box is checked in the Phase group box.
 - d. The **Fault** check box is checked in the Type group box.
 - e. The **Closed** and **Patched** check boxes are checked in the Status group box.

Note If you want to restrict the search to the hot fixes that were produced, check the **A hot fix was created** check box on the **Advanced Search Criteria II (Optional)** sheet.

2. On the **Advanced Search Criteria III (Optional)** sheet:
 - In the **Closed** list box of the Releases group box, select the release whose fixed PARs you want to locate (for example, the **6.3.0** list item).
3. Click the **Search** button.

Upgrading to JADE 6.3.06 from a JADE 6.2 or Earlier JADE 6.3 Release

This section covers the following topics.

- [Upgrading to JADE 6.3 from a Windows JADE 6.2 or Earlier JADE 6.3 Release](#)
 - [Running Two Windows Releases of JADE on the Same Workstation](#)
- [Upgrading to JADE 6.3 from a Linux JADE 6.2 or Earlier JADE 6.3 Release](#)
- [JADE Thin Client Upgrade](#)
- [Upgrading a Synchronized Database Environment \(SDE\)](#)
- [Upgrading an RPS Node from JADE 6.2 to 6.3](#)
- [Upgrade Validation](#)
- [Hot Fix Releases](#)

Upgrading to JADE 6.3 from a Windows JADE 6.2 or Earlier JADE 6.3 Release

You can upgrade from a 6.2 release to the 64-bit edition of JADE only if patch versioning has never been used in the JADE system. If patch versioning has ever been used, you must first upgrade to the 32-bit edition before you can upgrade to the 64-bit edition.

If you are upgrading to the 32-bit edition of JADE, the Microsoft Windows C++ 2005 Redistributable Package (x86) called **vc_redist_x86.exe** is required. If you are upgrading to the 64-bit edition, the Microsoft C++ 2008 SP1 Redistributable Package (x64) called **vc_x64Runtime.exe** is required. (These executables are supplied on the JADE distribution media.)

Note Installing this Microsoft redistributable package requires administration privileges. If possible, deploy this package to all workstations *before* upgrading to JADE 6.3, using the appropriate techniques that allow for privileged installations. If the required package is not already installed, it will be installed during the JADE installation.

The JADE Setup program enables you to upgrade your binary and database files to JADE 6.3 from a JADE 6.2 or earlier 6.3 release on Windows, by performing the following actions.

1. Ensure that your JADE environment is JADE release 6.2 or 6.3.
2. If you are upgrading from JADE 6.2, uninstall any **jadrap** database services (that is, JADE Remote Node Access services) that you have set up. (This is required because of changes in the underlying registry entries that are expected or created in JADE 6.3.)
3. If you are upgrading to JADE release 6.3 under Windows Server 2008 or Vista, ensure that you have the appropriate privileges or capabilities to install applications.

The configuration of Windows Server 2008 or Vista's User Account Control (UAC) and your current user account privileges may affect the behavior of the upgrade to JADE 6.3. For details about Windows Server 2008 or Vista UACs, standard user accounts, and administrator accounts, see:

<http://technet2.microsoft.com/WindowsVista/en/library/00d04415-2b2f-422c-b70e-b18ff918c2811033.msp?mfr=true>

[http://technet.microsoft.com/en-us/library/cc709691\(ws.10\).aspx](http://technet.microsoft.com/en-us/library/cc709691(ws.10).aspx)

4. Take a full backup copy of your existing JADE directories, first ensuring that your database is not in recovery mode.

Caution As roll-forward recovery of the installation and upgrade process is not supported, it is important that you backup your database *before* starting the JADE Setup process to install JADE 6.3 and upgrade your existing data.

5. To start the JADE Setup program, invoke the **setup.exe** program from the **Jade63** release medium or execute the executable program downloaded from the JADE Web site.
6. The Microsoft Windows C++ Redistributable Package is installed, if not already installed.
7. In the Welcome folder, click the **Next>** button to continue the upgrade process.
8. Read the entire software license agreement in the Software License Agreement folder and then click the **Yes** button to continue the installation.
9. In the Installation Type folder, select the **Feature Upgrade** option button, to specify that you want to upgrade an existing JADE release. By default, the **Fresh Copy** option is selected.
10. In the Setup Type folder, select the type of upgrade that you require. By default, the **Development** option is selected. If you do not want development files upgraded, select the **Application Runtime, Presentation Client, Jade Client, or SDS/RPS Database Server** option button, as required.

Note The **Custom** type applies only to a **Fresh Copy** installation type, and is not relevant when upgrading binary and database files. The **SDS/RPS Database Server** option applies only to the **Feature Upgrade** installation type.

11. In the Select Installation Folders folder, specify the locations of the JADE files that are to be upgraded.

The upgrade process defaults to the most-recently used JADE files, and displays these values in the **Install Directory**, **Executable Directory**, and **Database Directory** text boxes. The installation directory is most likely to be the root directory in which you installed JADE, unless you subsequently renamed the root directory or moved the files to another location.

If the locations are not as required, click the adjacent browse buttons (indicated by the ... ellipsis symbols) to display the common File Selection dialog that enables you to select the appropriate directories and files. By default, the **jade.ini** file located in your specified database directory is used.

If required, use the **JADE INI File** text box to specify a different valid fully qualified directory and name of the JADE initialization file; for example:

```
d:\mysys\jadetest\system\jadetest.ini
```

If Program Start folders are to be updated, specify the name of the folder in the **JADE Program Folder** text box. If you are unsure of the folder to be updated, click the adjacent **browse** button to display the common Folder selection dialog that enables you to select the folder.

The **Database Directory** text box enables you to explicitly specify the location in which the database (system) files are installed.

When installing on a non-Windows Server 2008 or Vista operating system or on Windows Server 2008 or Vista when the destination folder is not **\Program Files**, the database destination defaults to **system** under the install folder (for example, if you specify **c:\Jade63** in the **Install Directory** text box, the database directory defaults to **c:\Jade63\system**).

If the installation directory is a subdirectory of the programmatically determined location of **\Program Files** on Windows Server 2008 or Vista, the **\Program Files** portion of the install directory is replaced with the programmatically discovered location for the common application data directory (for example, if you specify **c:\Program Files\Jade63** in the **Install Directory** text box, the default database location is **c:\ProgramData\Jade63\system**).

The process checks whether the specified database directory is a valid system and that it is the correct ANSI or Unicode type.

12. The **JADE JIT Debugger** folder is displayed. The JADE Just-In-Time (JIT) debugger is required to reliably acquire a dump and crash log when an exception occurs in a system running with the Microsoft Visual C++ run time.

Note It is recommended that you install this on all machines hosting JADE nodes.

13. The Start Copying Files folder summarizing your upgrade options is displayed. If the selections displayed in the Start Copying Files folder are correct, click the **Next>** button. Alternatively, click the **<Back** button to modify your selections.
14. The Question dialog is displayed, advising you to ensure that you have taken a full backup of that database before you proceed with the upgrade process.

When you are sure that you are upgrading the correct system (and that it has been backed up), click the **Yes** button to start the upgrade process.
15. A warning message box is then displayed, advising you that Dynamic Link Libraries (DLLs) may need to be recompiled. Click the **OK** button, to continue.
16. A warning message may be displayed if the upgrade validation process has not completed. If so, check the **jadeupgrade.log** file for details about what needs to be modified in your user schemas to pass the validation and enable application execution.
17. When the upgrade is complete, the JADE Setup program informs you that the JADE Setup was successfully completed and that you can now view the **ReadMe.txt** file.

To view the **ReadMe.txt** file, ensure that the check box is checked (the default). The **ReadMe.txt** file is then displayed in a text editor (for example, Notepad). The **ReadMe.txt** file is a read-only text file installed in your JADE root directory that you can print or delete, if required. This file contains a reference to other JADE-related documents.

18. Click the **Finish** button to end the JADE upgrade process.
19. Install any **jadrapp** database services (that is, JADE Remote Node Access services) you had set up in JADE 6.2. (For details, see “[Running the Server Node as a Service](#)”, in the *JADE Remote Node Access Utility User’s Guide*.)

Caution As with any JADE release, you may need to recompile any external method Dynamic Link Libraries (DLLs) or external programs using the JADE Object Manager Application Programming Interfaces (APIs) with the new JADE **\Include** and **\Library** files before you attempt to run your upgraded JADE systems. (For details about the JADE Object Manager APIs, see Chapter 3 of the *JADE Object Manager Guide*.)

Some obsolete files are deleted from the JADE directories when upgrading from JADE 6.2. If you require these files for your JADE system, you must save them before you upgrade or restore them from the original JADE 6.2 release medium.

Running Two Windows Releases of JADE on the Same Workstation

You can have two releases of JADE installed on the same workstation, if the files are in different directories.

If ODBC is installed, only the last installation of the JADE ODBC driver is available from the ODBC Data Source Administrator.

Upgrading to JADE 6.3 from a Linux JADE 6.2 or Earlier JADE 6.3 Release

You can upgrade from JADE 6.2 to the 64-bit edition of JADE only if patch versioning has never been used in the JADE system. If patch versioning has ever been used, you must first upgrade to the 32-bit edition before you can upgrade to the 64-bit edition.

To upgrade from an existing JADE release to JADE release 6.3 on UNIX servers under SUSE Linux Enterprise Server 10.0 or Red Hat 5.0 or higher, perform the following actions.

1. Ensure that your JADE environment is JADE release 6.2 or earlier JADE 6.3 release.
2. Take a full backup copy of your existing JADE directories, first ensuring that your database is not in recovery mode.
3. Install the required JADE Red Hat Package Manager (RPM) file directly by using the standard Linux RPM install tools. This puts the files in the **/opt/jade** directory.

For details, see “Installing JADE”, in Chapter 3 or Chapter 4 of the *JADE Installation and Configuration Guide*.

4. To upgrade your JADE installation, use the **jadeinstall -U** parameter, as shown in the following example.

```
/opt/jade/sbin/jadeinstall -i /home/jade -U -v 6.3.03.000 --all
```

The parameter values are as follows.

- **-i <dir>** is the previously installed JADE directory
- **-v <version>** is the new JADE version to be installed
- **--all** indicates that all components previously installed into the directory specified in the **-i** parameter will be upgraded

The upgrade process copies over the new binaries and required system map files, resets timestamps, and performs any other steps necessary to complete the upgrade.

Caution As with any JADE release, you may need to recompile any external method libraries or external programs using the JADE Object Manager APIs with the new JADE **/include** and **/lib** files before you attempt to run your upgraded JADE systems. (For details about the JADE Object Manager APIs, see Chapter 3 of the *JADE Object Manager Guide*.)

For more details, see “Installing JADE on a UNIX Server under Linux” and “Parameters for the jadeinstall Command”, in Chapter 3 or Chapter 4 of the *JADE Installation and Configuration Guide*.

JADE Thin Client Upgrade

A JADE 6.3 presentation client upgrade:

- Rejects a presentation client upgrade from 5.2.08 or earlier. You must handle a presentation client download from JADE 6.1 by first upgrading JADE on the presentation client to release 6.0, 6.1, or 6.2.
- Cannot handle a reversion to JADE 6.1 or earlier.
- Rejects a reversion to JADE 6.0 or 6.1 if the JADE 6.0 or 6.1 application server attempts to download files to the **DownloadDirectory2** directory. The only reversion that is guaranteed is from JADE release 6.3 to JADE release 6.2.
- If you are upgrading presentation clients to JADE release 6.3 under Vista, ensure that you have the appropriate privileges or capabilities to install applications. The configuration of Vista's User Account Control (UAC) and your current user account privileges may affect the behavior of the upgrade to JADE 6.3.

For details about Vista UACs, standard user accounts, and administrator accounts, see:

<http://technet2.microsoft.com/WindowsVista/en/library/00d04415-2b2f-422c-b70e-b18ff918c2811033.msp?mfr=true>

When running JADE in thin client mode under Windows Vista or Windows XP, if the presentation client is installed:

- Under the **\Program Files** directory (or the **\Program Files (x86)** directory on a 64-bit machine with 32-bit JADE binaries), when an automatic presentation client upgrade occurs, you must have administrator rights or know the user name and password of a user with administrator rights. (This is a normal Microsoft requirement that updating of files under the **Program Files** directory requires administrative rights.)

If the Vista machine has had UAC disabled, the thin client upgrade will fail because of lack of permissions for standard users. For administration users, the necessary privileges are automatically granted so the upgrade will succeed.

If UAC is not disabled, administrative users are prompted with an **Allow** or a **Cancel** choice but standard users must know and supply the logon and password of a user with administrative privileges to enable the upgrade to succeed.

- Outside of the **\Program Files** directory (or the **\Program Files (x86)** directory), privilege elevation to perform an automatic thin client upgrade is not requested.

For more details, see Appendix B, "Upgrading Software on Presentation Clients", in the *JADE Thin Client Guide*.

Upgrading a 32-Bit Presentation Client Connecting to a 64-Bit Application Server

When a 32-bit presentation client connects to a 64-bit application server, the application server upgrades the version of the presentation client but it does not change the 32-bit to 64-bit type of the presentation client, because:

- The presentation client does not check to see if the operating system on which it is running is 64-bit-capable (and it would have to inform the application server about this).

- Any support libraries needed by the presentation client (for example, ActiveX control and automation libraries) would also have to be downloaded or already installed in the presentation client.

The 32-bit version of presentation client binaries must be installed on the 64-bit application server, in the `<jade-program-data-directory>/i686-msoft-win32-ansi/download/` directory structure.

If you require a 64-bit presentation client, you must manually install it. Once installed, it will automatically upgrade with 64-bit binaries.

Upgrading a Synchronized Database Environment (SDE)

To upgrade a Synchronized Database Service (SDS) installation from JADE release 6.2.12 or higher to JADE 6.3, perform the following actions.

1. Upgrade the primary system, as specified in earlier sections of this document.

Notes Upgrading a primary database causes a special **version check** trigger record to be written to the database journal to mark the boundary where conversion from JADE 6.2 to 6.3 occurred.

A 32-bit secondary cannot communicate with a 64-bit primary, or the reverse.

When upgrading an SDS database from a JADE 6.2 release, you must do so using the 32-bit software. When all SDS participants have been upgraded, you can then upgrade them to use 64-bit JADE software, if required.

2. Connect any JADE 6.2 native or RPS secondary database servers to the JADE 6.3 primary so that any remaining JADE 6.2 database journals are transferred and applied.

When the upgrade **version check** record is replayed, the following messages are recorded in the `jommsg.log` and tracking is halted.

```
SDS: Secondary upgrade version mismatch: tracking will now halt
SDS: Upgrade to the same software release level as the primary and
restart server
```

3. When database tracking halts at the upgrade trigger point, shut down the server and upgrade the secondary system by performing one of the following actions.
 - On Windows, use the automated InstallShield script provided in this release and select the **Feature Upgrade** option on the Installation Type dialog.

On the Setup Type dialog that is then displayed, select the **SDS/RPS Database Server** option.
 - Copy the:
 - i. Binary files (including `_sys*.bin` and `_jad*.bin` files)
 - ii. Monitor and dmpload map files
 - iii. Reset the timestamps

Upgrading an RPS Node from JADE 6.2 to 6.3

On upgrade from 6.2 to 6.3, the database type on the RPS node is set to **SQL Server 2000**, since SQL Server 2000 data types only were used in JADE 6.2.

If the RPS mapping database type was set to **SQL Server 2005** in the 6.2 system and you want the RPS mapping to use the SQL Server 2005 types, after the upgrade to 6.3 has completed, execute the following steps.

1. Start the RPS Manager on the RPS node.
2. Stop the **Datapump** application.
3. In the Configure RPS Node dialog, change the database type to **SQL Server 2005**.

This causes SQL scripts to be created and run to modify the columns to use the **SQL Server 2005** database types.

4. Start the **Datapump** application.

Upgrade Validation

During the upgrade process, a validation script is run to check the integrity of the upgraded system. Any user schema entities that conflict with system schema entities are logged as errors in the **jommsgn.log** file.

All errors must be corrected and validation re-run before user applications can be executed on the updated system. If the system is in the un-validated state, a message box is displayed when you log on to the JADE development environment, asking if validation should be re-run.

Hot Fix Releases

Hot fixes for JADE are released as binary files. To apply the hot fix:

1. Shut down the system.
2. Copy the hot fix system files into the appropriate directory.
3. Start up the system.

Caution You must apply all of the files contained in the hot fix at the same time. You do not need to reset the timestamps.

It is important to ensure that versions of JADE system files do not diverge from dependent binaries. Doing this ensures that dependent code files (JADE system files and libraries) are backed up and restored together. The default location of the JADE system files is the installation directory (that is, the **bin** directory for Windows and **\$JADEHOME/runtime** for Linux).

When it is necessary to restore a database from backup and perform recovery, you must avoid reverting to earlier JADE system file and binary versions. When restoring the binaries directory, ensure that it is from the latest backup.

Changes in JADE Release 6.3.06

This section contains changes in JADE release 6.3.06.

For details about changes in JADE releases 6.3.05 and 6.3.04, see “[Changes in JADE Release 6.3.05](#)” and “[Changes in JADE Release 6.3.04](#)”, later in this document. For details about changes in JADE release 6.3.03 (the first general release of JADE 6.3), see [RelInfo6303.pdf](#), in your JADE **documentation** directory.

Highlights in Release 6.3.06

The following table summarizes the new features in this release.

Feature	Description
Development environment enhancements	A number of customer suggestions have been implemented. See “ Development Environment Changes ” and “ Interfaces ”, later in this document, for details.
Transparent control skins	Transparent images are now supported in control skins.
Interpreter Output Viewer enhancements	New features now control the viewer window and save its contents to a file.
Web service statistics	A new getLastStatistics method retrieves useful Web service information.
Generate Web service consumer test stubs	Automatically generates test case classes and method stubs for Web service consumers.
SOAP format enhancements	Adds support for the RPC/Literal and Document/Literal Bare SOAP formats.
Partitioned files and secondary databases	Please read the important section “ SDS and Recovery Considerations ”, later in this document, for details of recovery considerations when using partitioned files in a Synchronized Database Environment (SDE).

Converting Pictures

The **Binary** primitive type **convertPicture** and **convertToFile** methods now raise exception 14015 (*File does not contain an image type that can be handled*) if the binary is not a valid image type.

Debugging an Application

The **Application** class now provides the **isBeingDebugged** method, which returns **true** if the application is currently being debugged using the JADE Debugger or **false** if the application is not being debugged. This method enables you to execute additional debugging code at run time when an application is being debugged.

See also “[\[FaultHandling\] Section](#)” under “JADE Initialization File”, later in this document.

Development Environment Changes

This section contains the JADE development environment changes in this release.

Adding a Method from a Free-Standing Method Editor Pane

The **New Jade Method**, **New External Method**, and **New Condition** commands in the Methods menu are now enabled when the focus is set to a free-standing method editor pane (for example, when you double-clicked on a method in the Methods List of the Class Browser).

When you select one of these commands, the corresponding method definition dialog (for example, the Jade Method Definition or Condition Definition dialog) is then opened. When you add the method, a new free-standing editor pane for that method is opened and focus is set to it so that you can define the new method to meet your requirements.

If you add multiple methods from the definition dialog, a free-standing editor pane is opened only for the last method that you added.

Adding a Schema

When the Add Schema dialog is now opened, you are required to enter only the schema name on the initial form of this dialog.

The default map file that is created is now based on the schema name and the map file for the global instance is now this default map file. (The Forms Management style and patch versioning also take on the same defaults as currently.)

If you want to change the default values, click the **Advanced** button so that the extended Add Schema dialog is displayed, enabling you to specify the values that you require before you click the **OK** button.

Class Browser

A class that is not defined in the schema currently being viewed is displayed in the editor pane of the Class Browser with a prefix of the name of the schema in which the class is defined; for example:

```
Class: ErewhonInvestmentsModelSchema::CommissionRate (2191)
```

Closing MDI Child Forms from the Windows Menu

The Menu Design form in the JADE Painter can now include the **Close All MDI Forms** command in the Window menu list on the **Menu Item** sheet. This command is included in the Window menu only when the **Window List?** check box is checked (which is the default value).

Note Existing JADE systems will not be affected. To add the new item to an existing Window list menu, uncheck the **Window List?** check box on the Menu Item sheet and then check it again.

At run time, if the user selects this menu item, all visible, enabled forms that do not have the **allowClose** property set to **false** are unloaded as if he or she has selected the **Close** button on each MDI child form. The **Form** class **queryUnload** and **unload** event methods are called, as normal. If any form rejects the unload (via the **cancel** parameter in the **queryUnload** event method), the unloading process will cease.

In addition, if the MDI frame has no visible and enabled MDI child windows displayed at run time, the **Arrange Icons**, **Cascade**, **Close All MDI Forms**, **New Window**, **Tile Horizontal**, **Tile Vertical**, and **Close All Windows** menu items are now disabled.

Creating Patch Files when Patch Versioning is Enabled

When patch versioning is enabled, the **Create Command File** check box on the **Schema Options** sheet of the Extract dialog continues to be checked by default, but it is no longer disabled so that you can now turn off the command file creation.

External Function and Library Extraction

The Functions menu, accessed from the External Functions Browser, now provides the **Extract** command. This displays the common Save As dialog, enabling you to specify the name and location of the extract file for the selected external function or library.

If the dialog is accessed when an external function library is selected, all external functions in that library are extracted to an **.scm** file, which has a prefix of **ExternalFunctions_***schema-name*. If the dialog is accessed when an external function is selected, that external function is extracted to a **.mth** file, which has a prefix of *schema-name_***external-function-name**.

The JADE development environment browser task and entity that you can define in the **jadeDevelopmentFunctionSelected** function **taskName** and **entityName** input parameters for this command is listed in the following table.

Task Name	Entity Name
mnuExtractFunction	<i>Schema-name</i>

Finding a Global Constant

When the Find Global Constant dialog is displayed, a maximum of 32,000 entries is now loaded into the list box.

If the specified text does not match any of the entries in the list and there are more than 32,000 entries, the rest of the entries are searched for the specified text. If a global constant is found, the list is cleared and repopulated with this entry and entries that follow this entry (again up to the 32, 000 limit). If fewer than 32, 000 entries, the behavior of this dialog is unchanged.

Listing Only the Most-Recently Accessed Methods in the History List

The **Browser** sheet of the Preferences or JADE Installation Preferences dialog now provides the **Consolidate History to show methods only** check box, which enables you to specify that you want to list only the most-recently accessed methods in the history list.

Type, property, and method entities are still added to the history list but accessing a method removes any existing type and property entities for a type. Subsequent type and property entries are added but a method entry again removes them.

By default, the most-recently accessed method, property, and type entities are displayed in the history list.

Patches Browser

The Patches menu, accessed from the Patches Browser, now provides the following commands.

- **Show Methods**, which displays the methods that were updated or added in the selected patch for the currently selected schema.

- **Show All Methods**, which displays the methods that were updated or added in the selected patch for all schemas.

The JADE development environment browser tasks and entities that you can define in the **jadeDevelopmentFunctionSelected** function **taskName** and **entityName** input parameters for these commands are listed in the following table.

Task Name	Entity Name
mnuShowAllMethods	<i>Schema-name</i>
mnuShowMethods	<i>Schema-name</i>

Text Templates

You can now add the **#patchNumber** variable template element on the **Text Templates** sheet of the Preferences or JADE Installation Preferences dialog (for example, to save a few seconds with each new method so that you can focus on the coding of that method when there is a requirement for all method comment sections to have a patch number).

When this template is inserted into a class, class constant, property, method, or interface, the value of the **#patchNumber** element is replaced by the current patch number of the user or it is blank if there is no current patch number; for example, your method template could look like the following.

```

/*****
Method Name:  #methodName
Developer:    #userId
Date created: #date
Patch Number: #patchNumber
*****/
vars

begin

end;
```

When a developer called Cedric adds a method to that method template, assuming that the patch number is 52593, it will look like the following.

```

testMethod();
/*****
Method Name:  testTemplate
Developer:    Cedric
Date Created: 03 March 2010
Patch Number: 52593
*****/
vars

begin

end;
```

See also “[Interfaces](#)”, later in this document.

Error Message

This section describes changed error message in this release.

14217 - This feature is not available in this environment

The existing exception 14217 can now also be raised if you attempt to clone instances of the **XamlObject**, **XamlDocument**, or **XamlManager** class or one of their subclasses, indicating that the feature cannot be implemented by the JADE Silverlight framework.

Exception Handling

The default exception dialog now:

- Sets the **Abort** button **cancel** property to **true** when running as a presentation client using **jade.exe**. As a result, pressing the ESC key generates an abort action.
- Displays labeled text boxes (that is, those displaying the error text and the source) as read-only text boxes, which enables you to copy text.

In addition, these text boxes now scroll horizontally if the text is too large for the control.

See also “[[FaultHandling](#)] Section” under “JADE Initialization File”, later in this document.

Executing Conditional Code

executeWhen conditional instruction flags are no longer required to be defined in the **JadeExecuteFlagCategory** global constant category but can be a **Boolean** global constant in any user-defined global constant category.

The value of the flag is not re-evaluated (that is, re-read from the initialization file) after the **clearMethodCache** method has been called. The current in-memory value is applied to each method as it is reloaded from the database.

The parameter values in the [JadeExecuteFlags] section of the JADE initialization file are loaded into memory the first time:

- An **executeWhen** instruction is encountered while loading a method
- The **Node::getExecuteFlagValue** method is called
- The **Node::setExecuteFlagValue** method is called.

The values are not read again until the node is restarted.

File Handling

This section contains the file handling changes in this release.

Browsing for Files

The file dialog that is displayed when you call the **FileFolder** class **browseForServerFolder** and **browserForAppServerFolder** methods now expands or collapses when the respective + or – icon image is clicked.

The form that is displayed when you call the **browseForFolder** method is a common File dialog whose functionality is not controlled by JADE. Clicking on these image icons has no effect when running JADE under some early versions of the Windows operating system. However, later versions of Windows (for example, Windows XP) *do* expand when the icons are clicked.

Maximum Size of Data on a Presentation Client

The maximum size of data that can be read or written by each read or write type statement when the value of the `TcpIpConnection` class `usePresentationClient` property or the `FileNode` class `usePresentationFileSystem` property is `true` is 2G bytes.

If the data exceeds this limit or the length is less than zero (**0**), exception 5047 (*Invalid record size*) is raised.

Forms and Controls

This section contains the changes to forms and controls in this release.

ComboBox Maximum Length

The implementation of the `maxLength` property for a `ComboBox` control has been altered to match the implementation on a Windows combo box.

With this change, the value of the `maxLength` property applies only when text is entered or altered by a user in the text box portion of a combo box. The value of the `maxLength` property no longer applies to the text set by logic or the text set by selecting an entry from the combo box.

The implementation of the `maxLength` property for the `TextBox` control is unchanged.

Control Skins

JADE has been enhanced to handle transparent images in a control skin.

JADE now forces the repaint of the parent or parents of the control, to achieve the correct layered drawing, when required, so that the presentation of the parent or parents is visible in the transparent parts of the image.

JPEG2000 Picture Format Support

With the exception of Compact JADE (which does not currently support this image type), JADE now supports JPEG2000 images.

The FreeImage library used by standard and Portable GUI JADE has been updated to support the handling of this image type.

The `Window` class now provides the `PictureType_Jpeg2000 (10)` class constant for this image type.

Interfaces

This section contains the interface change in this release.

Prefixing Stub Methods

When interface methods are automatically generated in implementing classes, you can now check the **Generate stub methods without prefix** check box in the Interface Options group box on the **Miscellaneous** sheet of the Preferences or JADE Installation Preferences dialog if you want new interface method names to be generated with the same name as the original interface method name, with no added prefix.

This check box is unchecked by default, indicating that interface methods are prefixed with a default value of **stub_** until you define another prefix in the **Generated stub method prefix** text box or you check this check box to set this value to null.

If you want to specify the method name prefix that is generated for all new implemented interface methods, specify the prefix that you require in the **Generated stub method prefix** text box on the **Miscellaneous** sheet of the Preferences or JADE Installation Preferences dialog.

Note These changes apply to new implemented interface methods only; that is, they do not affect existing implemented interface method names.

Specifying a Text Template for Interfaces

The **Text Templates** sheet of the Preferences or JADE Installation Preferences dialog now contains the **Interface** tab, which enables you to specify a template for the text of all new interfaces. For details, see “[Text Templates](#)” under “[Development Environment Changes](#)”, earlier in this document.

JADE Dynamic Objects

The following global constants are now defined in the **RootSchema**.

- JadeDynamicObjectNames::JStats_JadeBytesName ("JStatsJadeBytes")
- JadeDynamicObjectTypes::JStats_JadeBytesType (104)

These values are stored in the **name** and **type** attributes of the JADE dynamic object returned by the **JadeBytes** class **getStatistics** method.

JADE Initialization File

This section contains the JADE initialization file change in this release.

[FaultHandling] Section

The [FaultHandling] section now provides the **EnableDebugButton** parameter, which defaults to **true** and controls whether the **Debug** button is enabled on the default exception dialog. This parameter is read when the exception dialog is invoked.

Setting this parameter to **false** disables the Debug button on the dialog. When the application is running in thin client mode, the JADE initialization file on the application server is used to obtain the value of this parameter.

Note The **Debug** button is disabled automatically if the error is 1201 (*Kernel stack overflow*), because processing the **Debug** button would generate another kernel stack overflow or 3123 (*Access to objects in this secondary database is currently disabled*).

JADE Interpreter Output Viewer

This section contains the JADE Interpreter Output Viewer changes in this release.

Application::*clearWriteWindow* Method

The **Application** class now provides the **clearWriteWindow** method, which clears the contents of the JADE Interpreter Output Viewer if it is open. If the JADE Interpreter Output Viewer is not open, this method is ignored.

The **clearWriteWindow** method has the following signature.

```
clearWriteWindow();
```

Application::*closeWriteWindow* Method

The **Application** class now provides the **closeWriteWindow** method, which closes the JADE Interpreter Output Viewer if it is open. If the JADE Interpreter Output Viewer is not open, this method is ignored.

The **closeWriteWindow** method has the following signature.

```
closeWriteWindow();
```

Database Path Display

The JADE Interpreter Output Viewer can now include additional information in the title bar.

Use the **WindowTitle** parameter in the [JadeInterpreterOutputViewer] section of the JADE initialization file to specify the information that you want displayed.

If the value of the **WindowTitle** parameter is:

- Not empty and not **<default>**, the title is:

```
Jade Interpreter Output Viewer - WindowTitle-parameter-string-value
```

- Empty or **<default>** and the initiator is a standard client, the title is:

```
Jade Interpreter Output Viewer - path=database-path
```

- Empty or **<default>** and the initiator is a presentation client, the title is:

```
Jade Interpreter Output Viewer - AppServer=application-server  
AppServerPort=port App=application-name Schema=schema-name
```

If the **WindowTitle** parameter value is not used, the information is taken from the command line.

Restricting Font Selection

To restrict the fonts available for selection in the JADE Interpreter Output Viewer to monospaced (fixed pitch) fonts only, the JADE Interpreter Output Viewer now provides the **Font (Fixed Pitch)** command in the Options menu. When you select this command, the common Font dialog is then displayed, listing only fixed-pitch screen fonts (for example, Courier New).

The common Font dialog that is displayed when you select the Options menu **Font** command lists all screen fonts available for selection.

Saving the JADE Interpreter Output Viewer Contents

The JADE Interpreter Output Viewer now provides the **Save As** command in the File menu, which displays the common Set Save As File Name dialog in which you can specify the location and name of the log file to which the contents are saved. (This command is not available in a JADE portable GUI environment.)

JadeBytes::getStatistics Method

The **JadeBytes** class **getStatistics** method now defines and returns the **tailSegmentSize** attribute in the **JadeDynamicObject** parameter when invoked on an empty **JadeBytes** object, as shown in the following example.

```
---JStatsBytes(104)---
embeddedVector = true
entrySize = 1
length = 1000000
segmentSize = 262144
virtualSegments = 4
committedSegments = 2
tailSegmentLength = 237856
tailSegmentSize = 262185
```

JadeHTMLClass::generateHTMLForJadeTagsOnly Property

The **JadeHTMLClass** class now provides the Boolean **generateHTMLForJadeTagsOnly** property.

If this property is set to **true**, only the **JADE_TAG** tags will be processed when the HTML is generated for a page. All other tags will not be altered in any way.

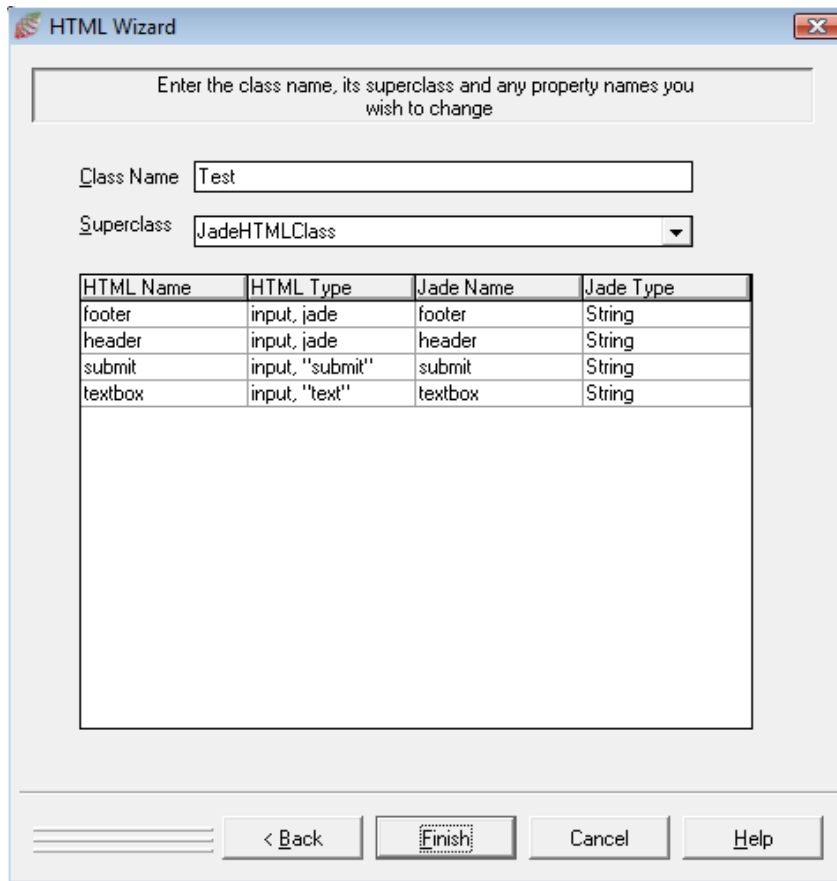
The default for this property is **false**, which will process all tags.

This feature changes the behavior of HTML document processing, as illustrated in the example in the following steps.

1. Use the HTML Wizard to import the following HTML.

```
<html>
<body>
<jade_tag name="header">
<form method="POST" action="http://localhost/jade/jadehttp.dll?S2">
<input type="text" name="textbox" value="">
<input type="submit" name="submit" value="OK">
</form>
<jade_tag name="footer">
</body>
</html>
```

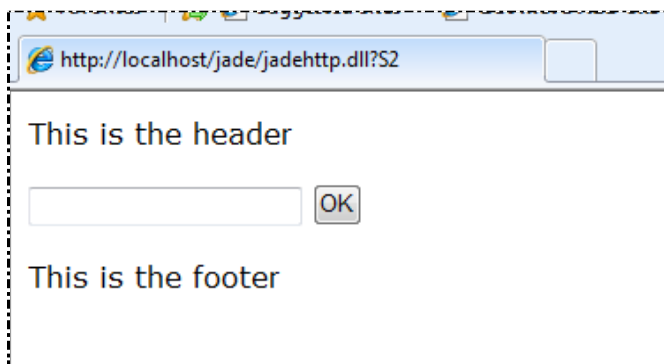
2. The HTML Wizard is then displayed as is shown in the following diagram.



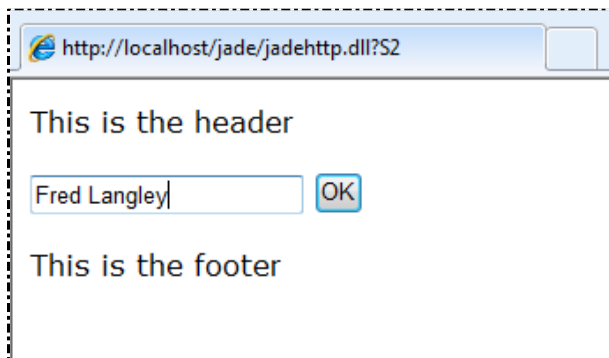
3. Add the following JADE code in the **updateValues** method for the **Test** class.

```
vars
begin
  header := "This is the header";
  footer := "This is the footer";
  return inheritMethod();
end;
```

4. The following diagram shows the Web browser when this page is opened.



5. Enter some data in the text box, as shown in the following diagram, and then click **OK**.



6. The page is now processed by Web application framework and the text box has the value **Fred Langley** displayed in it.
7. The Web application framework then calls the **updateValues** method again and generates the HTML for all tags including the **<input type=text ...>** tag, which is sent back to the Web browser.

Note Because the **<input type=text ...>** is also processed, the generated HTML contains the value of the corresponding property for this tag.

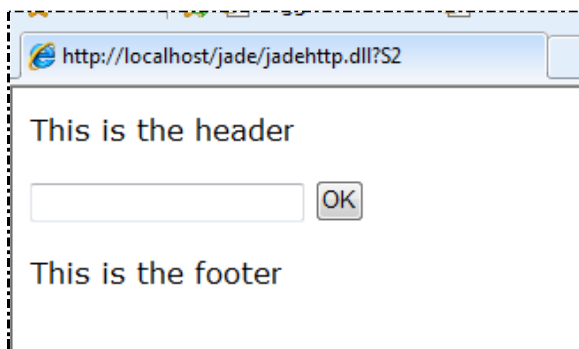
The Web browser is the same as that shown in step 5 of this example.

8. Add a **create** method with the following code to the **Test** class.

```
create() updating;
begin
    generateHTMLForJadeTagsOnly := true;
end;
```

9. Repeat steps 4 through 7 of this example.

The Web browser then looks like the following diagram.



The text box is now empty instead of having the text **Fred Langley** displayed in it, because now only the **JADE_TAG** tags are processed so the **<input type=text ...>** tag takes on its default value (which in this case is "", as can be seen in the HTML shown in step 1 of this example).

- One way to make this simple HTML work as before but still processing only `<jade_tag>` tags is to alter the HTML, as follows.

```
<html>
<body>
<jade_tag name="header">
<form method="POST" action="http://localhost/jade/jadehttp.dll?S2">
<input type=text name=textbox value="<jade_tag name=textBoxValue">">
<input type=submit name="submit" value="OK">
</form>
<jade_tag name="footer">
</body>
</html>
```

In this HTML, another `<jade_tag>` to represent the value of the text box has been added. Saving this text adds another String property `textBoxValue`, to represent the `JADE_TAG` tag.

- Change the `updateValues` method to populate the `textBoxValue` with the value of `textbox`, as shown in the following method.

```
updateValues(): Boolean updating;
vars
begin
    header := "This is the header";
    footer := "This is the footer";
    textBoxValue := textbox;
    return inheritMethod();
end;
```

- Running the application now has the same behavior as before the `generateHTMLForJadeTagsOnly` property was set to `true`, while generating HTML only for the `JADE_TAG` tags.

In general, any tag that is input-output is populated only with its initial value when the HTML message is generated, regardless of whether these values are in the message or changed in code. You therefore need to consider an approach like the one shown in this example if you need this information to be sent back to the Web browser.

Locking

Adding or removing an object to or from a transient or persistent and non-exclusive member key dictionary no longer share locks the object.

Relational Views

If you have a many-to-many relationship mapped in a relational view and the dictionary maps the same class to itself, you must recreate your relationship table by removing it and then adding it again by using the Relational View Wizard.

SDS and Recovery Considerations

Partitioned file structures and most meta data are replicated on SDS secondary databases.

You can change certain partition attributes such as location on the secondary in order to support a different storage strategy; for example, a tiered strategy on the primary and all partitions on the same default volume on the secondary.

Database tracking logic has been extended to support the replay of partitioned file operations, including file-level reorganization and compaction. Certain database file and file partition operations are replayed on SDS secondary databases but some are not.

The operations that are replayed in SDS are also reapplied by roll-forward recovery.

The following tables list database file and file partition operations that modify state. The second column indicates whether the operation is replayable; that is, the operation will be audited and reapplied by roll-forward recovery and SDS secondary replay. The third column indicates whether the operation can be executed on an SDS secondary database. In general, if the operation is replayable, it is not valid to execute the operation directly to an SDS secondary. Conversely, if an operation is not replayable, the operation can be executed on a secondary database, allowing the affected state to differ from the primary database.

The following table lists database file operations.

Operation	Replayable	Valid on SDS Secondary?
Set Partitioned	Yes	No
Create Partition	Yes	No
Set Partition Modulus	Yes	No
Freeze	No	Yes
Thaw	No	Yes
Mark Offline	No	Yes
Mark Online	No	Yes

The following table lists file partition operations.

Operation	Replayable	Valid on SDS Secondary?
Freeze	No	Yes
Thaw	No	Yes
MarkOffline	No	Yes
MarkOnline	No	Yes
Move	No	Yes
SetLabel	Yes	No
SetLocation	No	Yes

When considering use of partitioned database files in a Synchronized Database Environment (SDE):

- A file or partition that is not frozen on a primary can be frozen on a secondary.
 - If a file or partition is frozen on the secondary and updated on the primary, the file or partition must be located on writeable media to allow the database tracker to replay updates made to the primary version.
- It is possible and valid to have partitions:
 - Online on a secondary database that are offline on the primary.
 - This could be useful if secondary query applications need access to historical archived data that was taken offline on the production primary.

- Offline on a secondary database that are online on the primary, provided that the offline partitions are not updated on the primary database.

Ideally, the partitions should be frozen on the primary to protect against accidental updates. If the SDS tracker encounters updates that must be applied to an offline file or partition, tracking will halt. User intervention will be required to bring the required file or partition online before resuming tracking.

- When SDS is being used to implement a disaster recovery strategy, you should ensure that backup copies of partitions that are taken offline at the primary or secondary site can be restored at either site in the event of a disaster.

RPS Considerations

Partitioned file structures and meta data are replicated on full replica and mapped extent RPS databases. File partitioning is not applicable on a working set RPS database.

Stripping Source Code during a Batch Load

The batch JADE Schema Load utility (**jadloadb**) now enables you to strip the source code from all JADE methods in a schema.

You must specify the **executeSchema** command with a value of **RootSchema**, the **executeClass** command with a value of **Schema**, the **executeMethod** command with a value of **_removeSourceFromSchema**, and the **executeParam** command with the name of your user schema from whose methods you want source code stripped during the load process.

The syntax of the execute-related commands in the **jadloadb** command line is as follows.

```
jadloadb path=database-path
         ini=initialization-file-name
         executeSchema=RootSchema
         executeClass=Schema
         executeMethod=_removeSourceFromSchema
         executeParam=schema-name
```

The following example of the command line strips the source from all JADE methods in the **TestBaseSchema** schema.

```
jadloadb path=d:\jadesystems\jade63\system ini=c:\jade\jade.ini
executeSchema=RootSchema executeClass=Schema
executeMethod=_removeSourceFromSchema
executeParam=TestBaseSchema
```

An error is returned if the schema with the name specified in the **executeParam** parameter does not exist or if it is a system schema.

Terminating an Application

In earlier releases, when an application started another application and that initiation failed, the application starting the other application generated an exception based on that failure. If the initiated application called the **terminate** instruction in the **Global::getAndValidateUser** method during sign-on, error 1209 (*Application terminate request*) was reported, which caused the initiating application to also terminate.

From this release, if the initiated application calls the **terminate** instruction, no error is reported by the initiating application.

Thin Client Automatic Downloads

In earlier releases, if a presentation client connects to an application server and all of the required client binaries are not available to be downloaded, the presentation client could still connect and continue the session if the basic JADE release version of the presentation client binaries matched those of the application server.

From this release, the presentation client can no longer continue the session if all of the download files are not available. This change was required because the existing presentation client binaries may not be valid and consistent with the application server, even though the JADE versions match; for example, because of ANSI versus Unicode binaries, presentation client protocol changes, or client operating system changes.

Web Services

This section contains the Web services changes in this release.

Generating a Web Service Consumer Unit Test Class and Stub Methods

You can now generate a set of stub methods that can be used for Web service consumer unit testing.

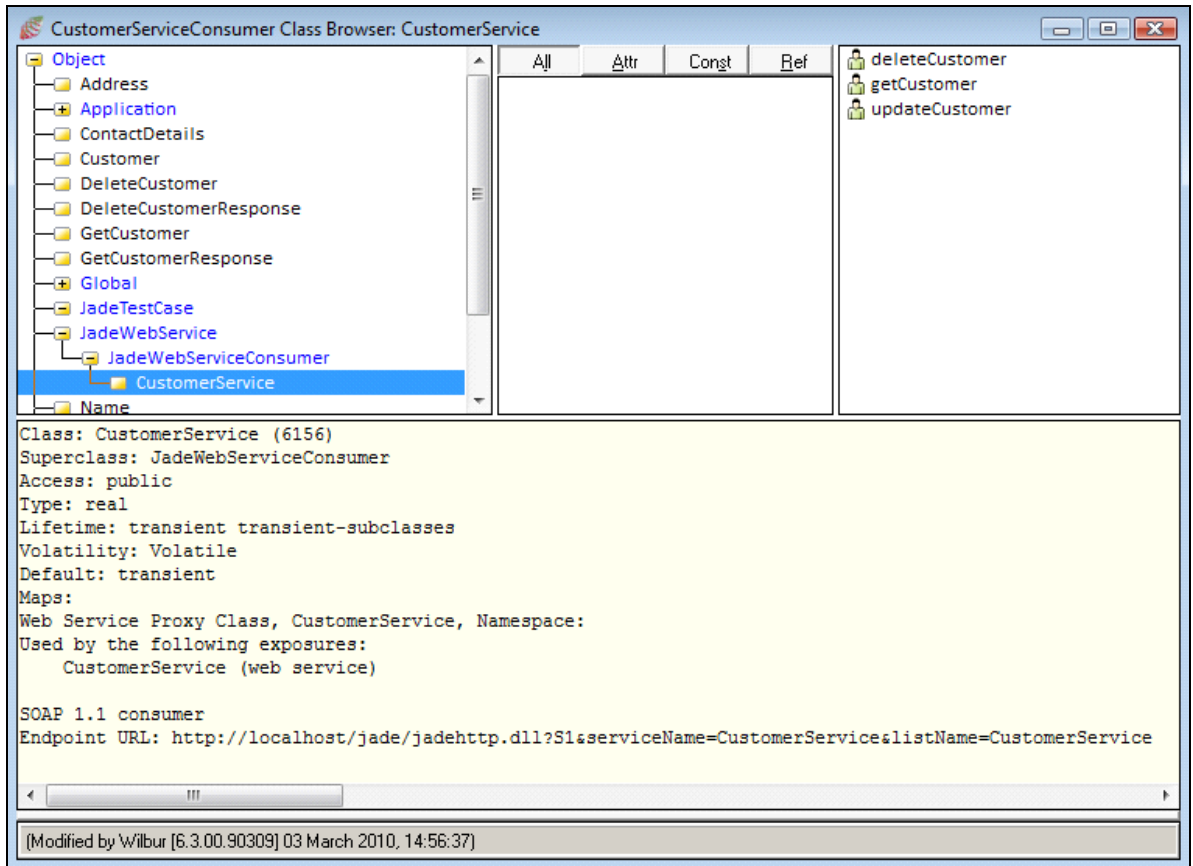
The Classes menu in the Class Browser provides the **Generate Test Case** command, which is enabled only when the selected class is a subclass of the **JadeWebServiceConsumer** class.

The JADE development environment browser task and entity that you can define in the **jadeDevelopmentFunctionSelected** function **taskName** and **entityName** input parameters for the **Generate Test Case** command is listed in the following table.

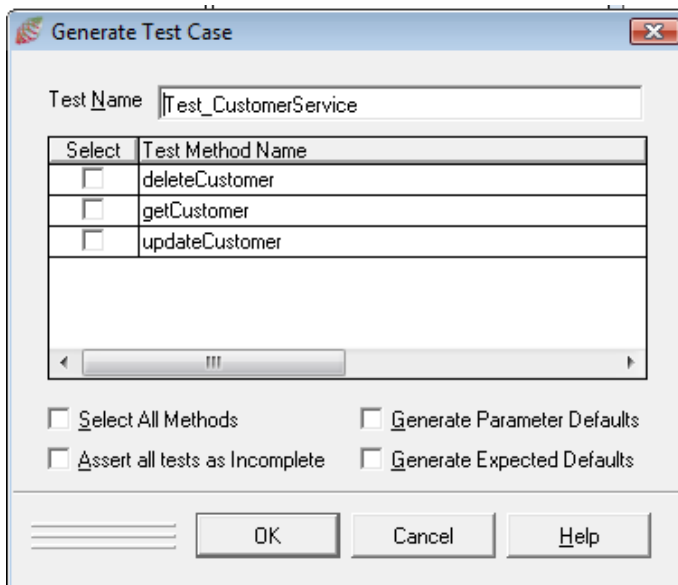
Task Name	Entity Name
mnuGenerateTestCase	<i>Schema-name</i>

For details, see “JADE Development Environment Security”, in Chapter 2 of the *JADE Object Manager Guide*.

In the description in this section, the sample Web service that is used is called **CustomerService**, which exposes the three methods shown in the following diagram.



The Generate Test Case dialog, shown in the following diagram, is displayed when you select the **Generate Test Case** command in the Classes menu.



The default class name of the **JadeTestCase** subclass that is created is the Web service consumer name (**CustomerService**, in this example) with the **Test_** prefix.

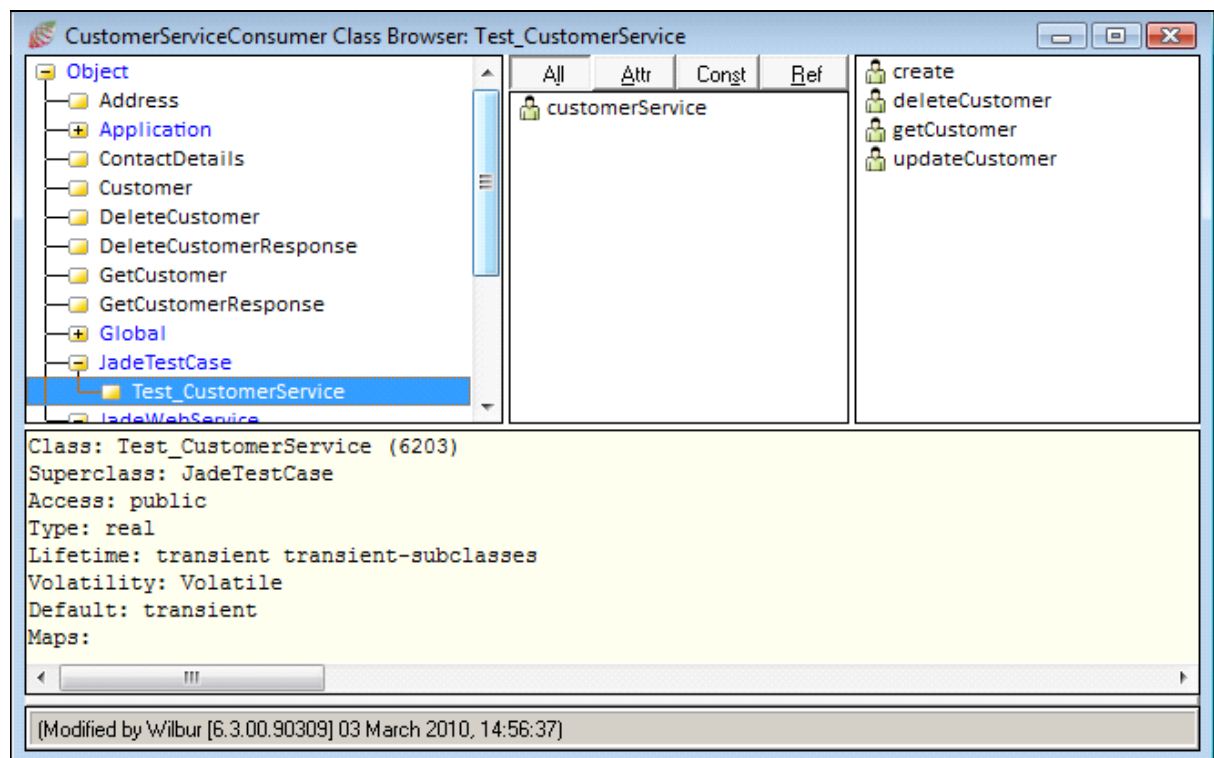
All of the methods that are Web service methods are displayed for selection in the Generate Test Case dialog. Each of the selected methods generates a method stub in the **JadeWebServiceConsumer** class subclass.

The generate options, available in check boxes on this dialog, are listed in the following table.

Option	Description
Select All Methods	Provides a quick way of selecting all of the methods
Generate Parameter Defaults	Generates stub code for the method parameters
Assert all tests as Incomplete	Adds an assert to the code to say test is not complete
Generate Expected Defaults	Generates stub code for the expected return value

For the example in this section, the name of the test has been left as **Test_CustomerService** and all of the check boxes have been checked.

When the **OK** button is clicked, the **JadeTestCase** subclass, shown in the following diagram, is then generated.



In addition to the stub methods for each of the Web service methods defined in the Web service, a **create** method and a **customerService** property that references the Web service are also generated.

The **customerService** property is initialized in the **create** method to the Web service from which the test was generated.

The code that was generated for the `getCustomer` method is shown in the following diagram.

```

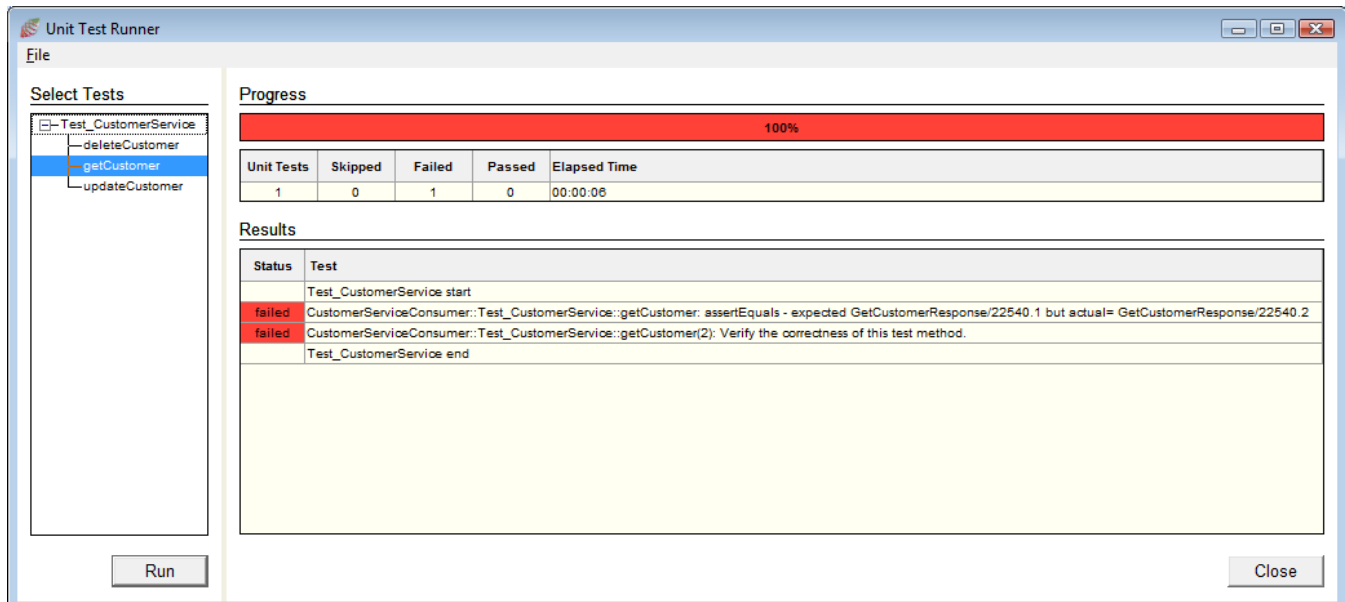
1 getCustomer() unitTest;
2
3 // Generated on 23 February 2010 by cnwcrs1
4
5 vars
6     actualValue: GetCustomerResponse;
7     expectedValue: GetCustomerResponse;
8     parameters : GetCustomer;
9     _getCustomerResult : Customer;
10 begin
11     // create the expected return value and initialise properties
12     create expectedValue transient;
13
14     // Create Customer instance and initialise primitive property values
15     create _getCustomerResult transient;
16     expectedValue.getCustomerResult := _getCustomerResult;
17     _getCustomerResult.address := null; // may need to create instance and set up property values
18     _getCustomerResult.contactDetails := null; // may need to create instance and set up property values
19     _getCustomerResult.customerNumber := 0;
20     _getCustomerResult.name := null; // may need to create instance and set up property values
21
22     // Initialise parameters to the appropriate value
23     create parameters transient;
24     parameters.customerNumber := 0;
25
26     // Call the web service method
27     actualValue := customerService.getCustomer(parameters);
28     // Test the returned value and the expected value
29     assertEquals(expectedValue, actualValue);
30     assert("Verify the correctness of this test method.");
31 end;
```

In this code:

- Lines 12 through 20 represent the expected defaults and are generated when the **Generate Expected Defaults** check box is checked.
- Lines 23 and 24 represent the parameter default values that are generated when the **Generate Parameter Defaults** check box is checked.
- Line 27 is the call to the Web service.
- Lines 29 and 30 are the default asserts. Line 30 is generated when the **Assert all tests as Incomplete** check box is checked.

Note This assert serves only as a reminder to you that the method needs to be changed.

As this is a stub method only, running this test without any modifications is not likely to work; for example, pressing F9 on the method in this example to run the test results in the failed results shown in the following diagram.



The first assert fails because the expected value does not match the return value.

The second assert always fails. When the test has been coded correctly, this assert should be removed. For example, if you want this test to check that the **customerNumber** property in the **Customer** object returned by the Web service is the same as the requested number, the following diagram shows the required method modifications.

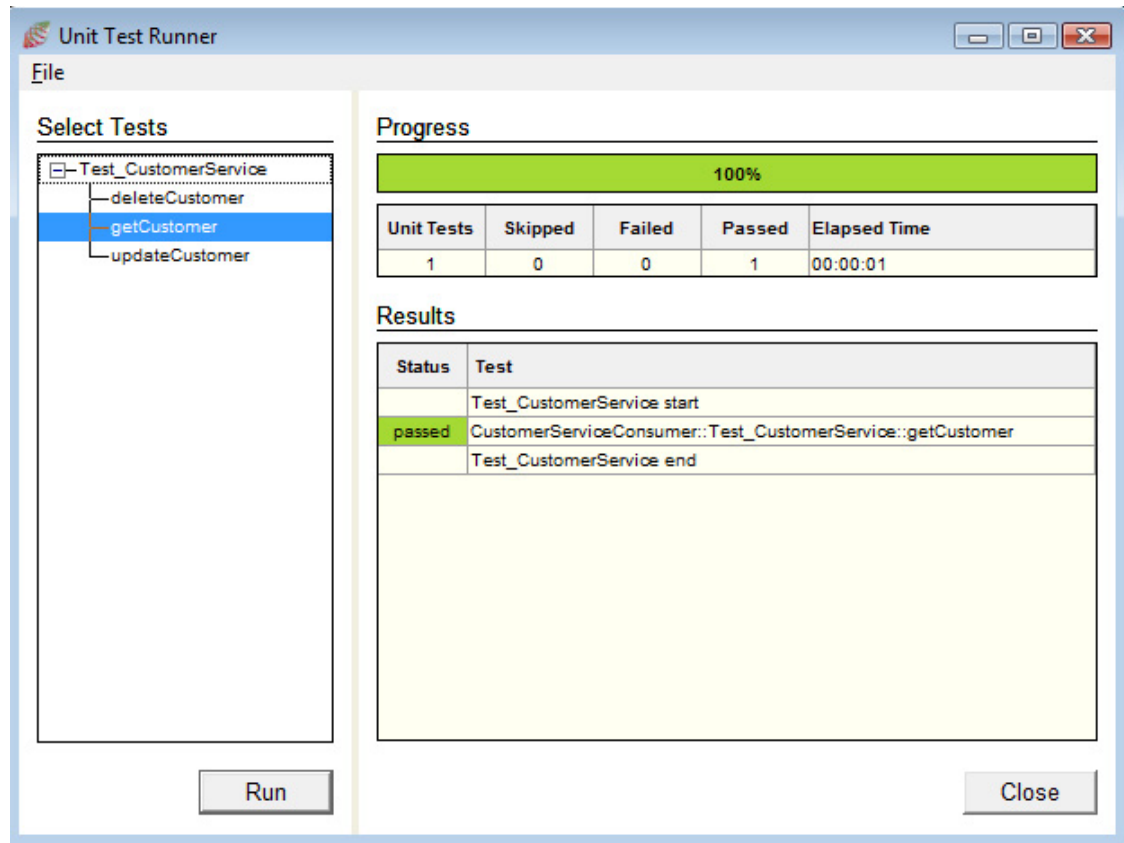
```

1 getCustomer() unitTest;
2
3 vars
4     actualValue: GetCustomerResponse;
5     expectedValue: GetCustomerResponse;
6     parameters : GetCustomer;
7     _getCustomerResult : Customer;
8 begin
9     // create the expected return value and initialise properties
10    create expectedValue transient;
11
12    // Create Customer instance and initialise primitive property values
13    create _getCustomerResult transient;
14    expectedValue.getCustomerResult := _getCustomerResult;
15    _getCustomerResult.address := null; // may need to create instance and set up property values
16    _getCustomerResult.contactDetails := null; // may need to create instance and set up property values
17    _getCustomerResult.customerNumber := 100;
18    _getCustomerResult.name := null; // may need to create instance and set up property values
19
20    // Initialise parameters to the appropriate value
21    create parameters transient;
22    parameters.customerNumber := 100;
23
24    // Call the web service method
25    actualValue := customerService.getCustomer(parameters);
26    // Test the returned value and the expected value
27    assertEquals(expectedValue.getCustomerResult.customerNumber, actualValue.getCustomerResult.customerNumber);
28 end;

```

Note that the last assert has been removed and that customer number 100 has been set up.

The result of running this modified test is shown in the following diagram.



Similarly, you can set up other expected values and compare these to the return values.

Note It makes sense to compare primitive type values only, as returned object instances will not match because they will always be different (transient) object instances.

JadeWebServiceProvider::getLastStatistics Method

The **JadeWebServiceProvider** class now provides the Boolean **getLastStatistics** method, which has the following signature.

```
getLastStatistics(headerOnly: Boolean): String;
```

This method returns an XML-formatted string that represents the information listed in the following table for the current request.

Statistic	Description
<queuedTime>	Time spent in queue
<requestTime>	Time taken to process the request
<webServiceTime>	Time spent in the web service method
<responseTime>	Time taken to generate the SOAP message and send the response
<requestSize>	Size of the request message
<responseSize>	Size of the response message

In addition, the information listed in the following table is returned when you set the **headerOnly** parameter to **false**.

Statistic	Description
<requestHeaders>	The HTTP headers that were received in the request
<soapRequest>	The SOAP message that was received
<soapResponse>	The SOAP message that was sent

To obtain all statistics for the request, you must call this method in your reimplemented **JadeWebServiceProvider** class **reply** method or in the destructor of the Web service.

Setting the value of the **headerOnly** to **true** returns a string similar to the following example.

```
<?xml version="1.0" encoding="utf-8"?>
<WebServiceStatistics>
  <queuedTime>5030</queuedTime>
  <requestTime>5</requestTime>
  <webServiceTime>4999</webServiceTime>
  <responseTime>4</responseTime>
  <requestSize>423</requestSize>
  <responseSize>387</responseSize>
</WebServiceStatistics>
```

Setting the value of the **headerOnly** parameter to **false** returns a string similar to the following example.

```
<?xml version="1.0" encoding="utf-8"?>
<WebServiceStatistics>
  <queuedTime>5016</queuedTime>
  <requestTime>5</requestTime>
  <webServiceTime>5000</webServiceTime>
  <responseTime>3</responseTime>
  <requestSize>423</requestSize>
  <responseSize>387</responseSize>
  <requestHeaders><![CDATA[Cache-Control: no-cache
Connection: Keep-Alive
Content-Length: 423
Content-Type: text/xml; charset=utf-8
Accept: text/plain, text/html, text/xml
Host: localhost
User-Agent: Jade/6.3.00
SOAPAction: "urn:JadeWebServices/Mine/helloWorld"
]]></requestHeaders>
  <soapRequest><![CDATA[<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:tns="urn:JadeWebServices/Mine/"
xmlns:s1="urn:JadeWebServices/Mine/">
  <soap:Body>
    <s1:helloWorld>
      <s1:user/>
      <s1:date>1900-01-01</s1:date>
    </s1:helloWorld>
  </soap:Body>
</soap:Envelope>
]]></soapRequest>
  <soapResponse><![CDATA[<?xml version="1.0" encoding="utf-8"?>
```

```

<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <helloWorldResponse xmlns="urn:JadeWebServices/Mine/">
      <helloWorldResult>true</helloWorldResult>
    </helloWorldResponse>
  </soap:Body>
</soap:Envelope>
]]></soapResponse>
</WebServiceStatistics>

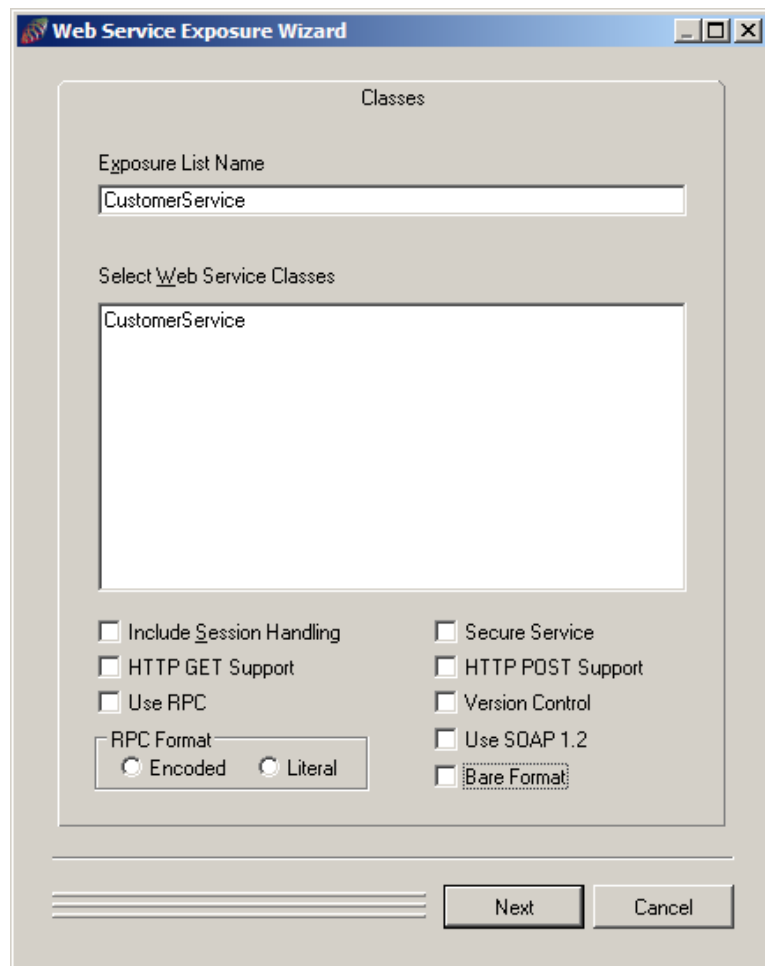
```

Support for the Document/Literal Bare SOAP Format

The JADE Web service framework now supports both the **Document/Literal Wrapped** and **Document/Literal Bare** Simple Object Access Protocol (SOAP) formats. In earlier releases, the wrapped format only was supported.

Support for the **Document/Literal Bare** format enables you to create a Web service provider that uses the **Document/Literal Bare** SOAP format. In addition, you can now import a JADE or external Web Service Definition Language (WSDL) that is in **Document/Literal Bare** format.

To create a provider that is in **Document/Literal Bare** format, the Web Service Exposure Wizard now provides the **Bare Format** check box, as shown in the following diagram.



Checking the **Bare Format** check box for the Web service exposure and using this exposure to generate a WSDL generates a WSDL in the **Document/Literal Bare** format.

Note You cannot use the **Document/Literal Bare** format if you select classes and properties to be exposed that will cause the references to be circular. When defining the classes and properties, you are warned of this, if this is the case.

There is no change to Web service consumer. The WSDL generated from this definition contains all of the information required for the import.

Support for the RPC/Literal SOAP Format

The JADE Web service framework now enables you to create a Web service provider that uses the **RPC/Literal** SOAP format. In addition, you can import a JADE or external WSDL that is in **RPC/Literal** format.

The Web Service Exposure Wizard now provides the **Use RPC** check box and the associated **RPC Format** group box, which enable you to create a Web service provider that is in **RPC/Literal** format when defining the exposure. For an example of this dialog, see “[Support for the Document/Literal Bare SOAP Format](#)”, in the previous section.

When you check the **Use RPC** check box, you can then select the **Literal** option button if you do not want the default **Encoded** option. Selecting the **Literal** option button enables you to set up a Web service exposure in the **RPC/Literal** format.

There is no change to the Web service consumer. The WSDL generated from this definition contains all of the information required for the import.

C-Level Application Programming Interface (API) Enhancement

JADE now provides C-level API calls that help C programmers to write external methods and functions to obtain JADE initialization file information, directory information, and to convert JADE characters to an ANSI or Unicode string. This enables consistent behavior between the C-level code and the JADE runtime.

Function prototypes for these calls are defined in the **joscalls.h** header in the **include** directory on the JADE release medium. The C functions return an **int**, which will be zero (**0**) indicating success or a JADE object manager error number.

For details, see the following subsections.

Getting the Name of the Initialization File

The **josIniFileGetFileName** call, shown in the following example, obtains the name of the JADE initialization file.

```
int josIniFileGetFileName(Character fileName[],
                          Size      size);
```

The parameters for this call are listed in the following table.

Parameter	Description
fileName	Name of the JADE initialization file
size	Size of the buffer to hold the JADE initialization file, defined in characters

Getting a Boolean Value from the Initialization File

The `josIniFileGetBoolean` call, shown in the following example, obtains the current value of a **Boolean** key in the specified section of the JADE initialization file.

```
int josIniFileGetBoolean(const Character * pIniFileName,
                        const Character * pSection,
                        const Character * pKey,
                        bool             bDefault,
                        bool &          bValue);
```

The parameters for this call are listed in the following table.

Parameter	Description
<code>pIniFileName</code>	Name of the JADE initialization file
<code>pSection</code>	Name of the section in the JADE initialization file
<code>pKey</code>	Name of the Boolean key in the JADE initialization file section
<code>bDefault</code>	Boolean default value for the specified key if the key cannot be found in the initialization file
<code>bValue</code>	Requested Boolean value from the initialization file

Getting a Signed Integer Value from the Initialization File

The `josIniFileGetSInteger` call, shown in the following example, obtains the current value of a signed **Integer64** key in the specified section of the JADE initialization file.

```
int josIniFileGetSInteger(const Character * pIniFileName,
                          const Character * pSection,
                          const Character * pKey,
                          Integer64       sDefault,
                          Integer64 &    sValue);
```

The parameters for this call are listed in the following table.

Parameter	Description
<code>pIniFileName</code>	Name of the JADE initialization file
<code>pSection</code>	Name of the section in the JADE initialization file
<code>pKey</code>	Name of the Integer64 key in the JADE initialization file section
<code>sDefault</code>	Integer64 default value for the specified key if the key cannot be found in the initialization file
<code>sValue</code>	Requested Integer64 value from the initialization file

Getting an Unsigned Integer Value from the Initialization File

The `josIniFileGetUInteger` call, shown in the following example, obtains the current value of an unsigned **Integer64** key in the specified section of the JADE initialization file.

```
int josIniFileGetUInteger(const Character * pIniFileName,
                          const Character * pSection,
                          const Character * pKey,
                          UInteger64      uDefault,
                          UInteger64 &    uValue);
```

The parameters for this call are listed in the following table.

Parameter	Description
pIniFileName	Name of the JADE initialization file
pSection	Name of the section in the JADE initialization file
pKey	Name of the Integer64 key in the JADE initialization file section
uDefault	Unsigned Integer64 default value for the specified key if the key cannot be found in the initialization file
uValue	Requested unsigned Integer64 value from the initialization file

Getting a String Value from the Initialization File

The **josIniFileGetString** call, shown in the following example, obtains the current value of the **String** key in the specified section of the JADE initialization file.

```
int josIniFileGetString(const Character * pIniFileName,
                       const Character * pSection,
                       const Character * pKey,
                       const Character * pDefault,
                       Character      * pValue,
                       Size            size);
```

The parameters for this call are listed in the following table.

Parameter	Description
pIniFileName	Name of the JADE initialization file
pSection	Name of the section in the JADE initialization file
pKey	Name of the String key in the JADE initialization file section
pDefault	String default value for the specified key if the key cannot be found in the initialization file
pValue	Address of a Character buffer into which the requested string value from the initialization file is placed
size	Size of the Character buffer to which the pValue parameter points, defined in characters

Setting a Boolean Value in the Initialization File

The **josIniFileSetBoolean** call, shown in the following example, sets the **Boolean** key value in the specified section of the JADE initialization file.

```
int josIniFileSetBoolean(const Character * pIniFileName,
                        const Character * pSection,
                        const Character * pKey,
                        bool             bValue);
```

The parameters for this call are listed in the following table.

Parameter	Description
pIniFileName	Name of the JADE initialization file
pSection	Name of the section in the JADE initialization file
pKey	Name of the Boolean key in the JADE initialization file section
bValue	Boolean value that is written to the specified key and section of the JADE initialization file

Setting a Signed Integer Value in the Initialization File

The `josIniFileSetSInteger` call, shown in the following example, sets the **Integer64** key value in the specified section of the JADE initialization file.

```
int josIniFileSetSInteger(const Character * pIniFileName,
                        const Character * pSection,
                        const Character * pKey,
                        bool bMultipliers,
                        Integer64 sValue);
```

The parameters for this call are listed in the following table.

Parameter	Description
pIniFileName	Name of the JADE initialization file.
pSection	Name of the section in the JADE initialization file.
pKey	Name of the Integer64 key in the JADE initialization file section.
bMultipliers	If true , the sValue parameter is checked to see if it is an exact multiplier, if possible, and the value is written with the appropriate multiplier suffix. Multipliers are case-insensitive K , M , or G <i>prefix multipliers</i> . (For details, see “Handling of Parameter Values”, in the JADE <i>Initialization File Reference</i> .)
sValue	Integer64 value that is written to the specified key and section of the JADE initialization file.

Setting an Unsigned Integer Value in the Initialization File

The `josIniFileSetUInteger` call, shown in the following example, sets the unsigned **Integer64** key value in the specified section in the JADE initialization file.

```
int josIniFileSetSInteger(const Character * pIniFileName,
                        const Character * pSection,
                        const Character * pKey,
                        bool bMultipliers, UInteger64 uValue);
```

The parameters for this call are listed in the following table.

Parameter	Description
pIniFileName	Name of the JADE initialization file.
pSection	Name of the section in the JADE initialization file.
pKey	Name of the key in the JADE initialization file section.

Parameter	Description
bMultipliers	If true , the uValue parameter is checked to see if it is an exact multiplier, if possible, and the value is written with the appropriate multiplier suffix. Multipliers are case-insensitive K , M , or G <i>prefix multipliers</i> . (For details, see “Handling of Parameter Values”, in the JADE <i>Initialization File Reference</i> .)
uValue	Unsigned Integer64 value that is written to the specified key and section of the JADE initialization file.

Setting a String Value in the Initialization File

The **josIniFileSetString** call, shown in the following example, sets the **String** key value in the specified section of the JADE initialization file.

```
int josIniFileSetString(const Character * pIniFileName,
                      const Character * pSection,
                      const Character * pKey,
                      const Character * pValue);
```

The parameters for this call are listed in the following table.

Parameter	Description
pIniFileName	Name of the JADE initialization file
pSection	Name of the section in the JADE initialization file
pKey	Name of the String key in the section in the JADE initialization file
pValue	String value that is written to the specified key and section of the JADE initialization file

Getting the JADE HOME Directory

The **josGetDirectoryJade** call, shown in the following example, obtains the name of the JADE HOME directory, which is the parent directory of the installation directory (for example, if your installation directory is **Jade\bin**, your JADE HOME directory is **Jade**).

```
int josGetDirectoryJade(Character pathName[],
                      Size      size);
```

The parameters for this call are listed in the following table.

Parameter	Description
pathName	Buffer to receive the name of the JADE HOME directory
size	Size of the pathName parameter buffer, defined in characters

Getting the JADE Installation Directory

The **josGetDirectoryJadeBin** call, shown in the following example, obtains the name of the JADE installation directory (commonly known as the **bin** directory) for Windows or the **\$JADEHOME/bin** directory for Linux.

```
int josGetDirectoryJadeBin(Character pathName[],
                          Size      size);
```

The parameters for this call are listed in the following table.

Parameter	Description
pathName	Buffer to receive the name of the directory where the executable of the current executing program is located; that is, the directory in which the JADE binaries are installed
size	Size of the pathName parameter buffer, defined in characters

Getting the JADE Lib Directory

The **josGetDirectoryJadeLib** call, shown in the following example, obtains the name of the directory in which shared objects and libraries are installed (commonly known as the **bin** directory) for Windows or the **\$JADEHOME/lib** directory for Linux.

```
int josGetDirectoryJadeLib(Character pathName[],
                          Size      size);
```

The parameters for this call are listed in the following table.

Parameter	Description
pathName	Buffer to receive the name of the directory where the shared objects and libraries of the current executing program are located
size	Size of the pathName parameter buffer, defined in characters

Getting the JADE Temp Directory

The **josGetDirectoryJadeTemp** call, shown in the following example, obtains the name of the **temp** directory; for example, the directory to which files are extracted or installation files are downloaded. This directory is often **Jade\temp** for Windows or the **\$JADEHOME/tmp** directory for Linux.

```
int josGetDirectoryJadeTemp(Character pathName[],
                             Size      size);
```

The parameters for this call are listed in the following table.

Parameter	Description
pathName	Buffer to receive the name of the JADE temp directory
size	Size of the pathName parameter buffer, defined in characters

Converting a JADE Character to an ANSI Value

The **josCharacterToAnsi** call, shown in the following example, converts a JADE **Character** type to an ANSI value.

```
int josCharacterToAnsi(Character * pSource,
                      char      target[],
                      Size      sizeOfTargetInChar);
```

The parameters for this call are listed in the following table.

Parameter	Description
pSource	String of JADE characters that are to be converted
target	Location of the buffer to receive the converted string
sizeofTargetInChar	Size of the target buffer, defined in ANSI characters

Converting a JADE Character to a Unicode Value

The `josCharacterToUnicode` call, shown in the following example, converts a JADE `Character` type to a Unicode value using the UTF 16 (for Windows) or UTF 32 (for Linux) wide-character encoding routine.

```
int josCharacterToUnicode(Character * pSource,
                        wchar_t   target[],
                        Size      sizeofTargetInWChar);
```

The parameters for this call are listed in the following table.

Parameter	Description
pSource	String of JADE characters that is to be converted
target	Location of the buffer to receive the converted string
sizeofTargetInWChar	Size of the target buffer, defined in <code>wchar_t</code> characters

Changes in JADE Release 6.3.05

This section contains changes in JADE release 6.3.05.

For details about changes in JADE release 6.3.04, see “[Changes in JADE Release 6.3.04](#)”, later in this document. For details about changes in JADE release 6.3.03 (the first general release of JADE 6.3), see [RelInfo6303.pdf](#), in your JADE **documentation** directory.

Highlights in Release 6.3.05

The following table summarizes the new features in this release.

Feature	Description
Delta database	Allows applications to make non-permanent changes to a database when it is in read-only mode and to easily discard these changes
Drawing capture on Picture controls	New methods let you draw on a Picture control by capturing mouse events, which is very useful for recording signatures, for example
System sequence numbers	Generate a series of consecutive unique numbers without requiring object locks
.NET import handles boxed primitives	Greater interoperability with .NET components via support for boxed primitives
Combo box lists can now display 20 entries	The maximum number of entries in the list portion of a combo box has been increased from 8 to 20
Base64 methods	The RootSchema now provides methods to encode and decode base64 strings
Translatable strings in HTML documents	A new JADE_TAG allows you to use translatable strings in HTML documents
Merge latest partial and full backups	Create sophisticated backup and recovery applications by merging your latest partial and full backups
Look up a control by name	A new Form class method allows you to find a control on a form by name
Development environment editor splitter	The JADE development environment (IDE) editor pane can now be split into two horizontal views

.NET Component Importing

The **JadeDotNetType** class now provides the methods documented in the following subsections, which enable you to store JADE primitive types and retrieve them via .NET objects. (In .NET, the process of storing a primitive type in an object is called boxing, so JADE uses a similar name for its methods.)

JadeDotNetType::createBoxedPrimitive Method

The **JadeDotNetType** class **createBoxedPrimitive** method, which has the following signature, creates a .NET object from a JADE primitive type.

```
createBoxedPrimitive (value: Any);
```

Note The **value** parameter can be a primitive type only; it cannot be a class.

You can call this method, for example, to set the chart type (which is an **Object** .NET type), as shown in the following code fragment.

```
vars
    boxed : JadeDotNetType;
begin
    create boxed;
    boxed.createBoxedPrimitive
        (DotnetCHARTING_WinForms.SeriesType_AreaLine);
    chart1.defaultSeries.type := boxed;
    ...
```

JadeDotNetType::getBoxedPrimitive Method

The **JadeDotNetType** class **getBoxedPrimitive** method, which has the following signature, extracts a primitive type from a .NET object.

```
getBoxedPrimitive(): Any
```

Non-Default Constructors on Imported .NET Components

JADE provides the **JadeDotNetType::createDotNetObject** method to create the underlying .NET object for a component whose definition has been imported into JADE. Internally, JADE creates the .NET object by calling the default constructor.

A default constructor is a constructor that does not take any parameters. A non-default constructor is a constructor with parameters. In JADE versions prior to this release, you could not access .NET non-default constructors.

From this release, during the import of a .NET component, JADE now creates methods in the imported proxy class to provide access to the non-default constructors. The names given to the imported non-default constructor methods are of the form **createDotNetObject_n**, where *n* is a number that ensures the name is unique within JADE.

You can call these new methods in place of the existing **createDotNetObject** method. To use a non-default constructor, create the .NET component.

Batch JADE Database Utility

Since the introduction of the capability to freeze files and partitions, and to exclude them in a backup operation, some sophisticated backup applications have sought the ability to merge such a partial backup with the prior full backup of the environment, thus simplifying operational issues restoring the environment from backup.

The batch JADE Database utility (**jdbutilb**) **makeBackupinfo** command, available in this release, applies the latest partial backup (delta) **backupinfo** file to the last full (base) backup **backupinfo** file, creating a full backup **backupinfo** file.

File and partition backup information in the delta **backupinfo** file is copied to the output **backupinfo** file. File and partition information that is absent from the delta **backupinfo** file is copied from the base **backupinfo** file to the output **backupinfo** file.

Note It is the responsibility of the backup application to relocate necessary files.

You can use the new **listBackupinfoFileNames** command to retrieve the names of database files, partitions, Unstructured Data Resource (UDR), and single UDR files from the **backupinfo** file. The name list is useful for managing clean-up at the full backup location; for example, detecting when a file has been removed from an environment.

The batch JADE Database utility now provides the following commands. (See also “[JADE Database Batch Utility \(jdbutilb\) Command](#)” under “[Delta Databases](#)”, later in this document.)

listBackupinfoFileNames Command

The **listBackupinfoFileNames** command lists the names of all files in the **backupinfo** file located in the directory specified by the **path** parameter.

makeBackupinfo Command

The **makeBackupinfo** command, which has the following syntax, uses a partial **backupinfo** file as the master file and merges the base **backupinfo** file to create a new **backupinfo** file.

```
makeBackupinfo baseDir=directory deltaDir=directory outputDir=directory
```

New Error Messages

The new error messages that can be raised when performing batch JADE Database utility actions are as follows.

- 3173 - File not backed up

This error occurs if:

- The file or partition is not backed up in the specified location when using the **restoreFile** or **restoreFilePartition** command.
- An error condition was detected when copying the file or partition information from the base **backupinfo** file using the **makeBackupinfo** command.

When a file or partition is not backed up in a delta, the file cannot have been modified between the base backup and the delta backup. When modification is detected, the 3173 error occurs because the file or partition should have been backed up. Current and base **backupinfo** file update timestamps must match and the single UDR file lists must match. For more-specific information, see the **jommsg** log file.

- 3174 - Base backupinfo is newer than delta backupinfo

You can create a new **backupinfo** file only by merging the current delta (partial) backup information with the previous base (full) backup information.

- 3175 - Partitioned status of file differs in base and delta backupinfo files

When a file or partition is not backed up in a delta, the file cannot have been modified between the base backup and the delta backup, and a partitioning change constitutes a modification, so the file should have been backed up; that is, included in the delta.

- 3176 - Base backupinfo is not a full backup

You can create a new **backupinfo** file only by merging the current delta (partial) backup information with the previous base (full) backup information.

Code Coverage

The code coverage changes in this release are as follows.

- The **Covered Seconds** column in the table at the top right of the Code Coverage Results Browser has now been excluded from the code coverage analysis data, as the values were not correctly calculated.
- The table at the top-right of the Code Coverage Results Browser now displays the following information for the entity selected in the **Entity** list box at the top left.

Column	Description
% of JADE Methods Executed	Percentage of the number of JADE methods defined on the class or schema that were executed. The entry is empty if the selected entity list item is not a class or schema. You must manually request the percentage value for a schema (by selecting the Calculate Total Methods Executed Percent for Schema command from the File menu).
Total Blocks in executed methods	Total number of blocks that are available for execution by the executed methods for the selected entity. (Each method consists of a number of executable blocks.) Note that for schema, class, and primitive type entries, this does not include the number of blocks for methods that have not yet been executed.
Covered Blocks	Number of blocks that have been executed at least once. If a single block is executed multiple times, it is still counted as a single covered block.
Covered Blocks %	Number of covered blocks as a percentage of the total blocks. Note that for schema, class, and primitive type entries, this does not include the number of blocks for methods that have not yet been executed.
Not Covered Blocks	Number of blocks not executed at all (that is, total blocks minus the number of covered blocks).
Not Covered Blocks %	Number of blocks not executed as a percentage of the total blocks.
Executed Count	Total number of times a method was executed.

- The **Load** command in the File menu now provides the **Load and Merge Coverage for Methods** and **Load and Replace Coverage for Methods** commands.

These commands display a common dialog, which enables you to select a code coverage output file (or files) to load. The information in the file is merged with the currently loaded information, if any.

When merging a file (that is, loading a file into a non-empty session), coverage information may be loaded for methods for which there is already coverage information (in the current session).

In such cases, method editions are used to determine which has the latest coverage information; that is, the current session or the file being loaded.

- When replacing coverage information, if the method edition in the currently loaded information is greater than the method edition in the file, the current session is assumed to already have the latest coverage information and the data in the file (for this method only) is ignored.

If the method edition in the file is greater than or equal to the edition of the method in the current session, the file is assumed to contain later information, so the current session information (for this method only) is discarded and recreated from the file being loaded.

- When merging coverage information, if the current session already has coverage information for a method and the method editions are the same, the new coverage information for the method is merged with the existing information.
- The **Save Schema Methods Not in Coverage as CSV** command in the File menu is enabled only when a schema is selected in the **Entity** list box.

A common dialog is then displayed, which prompts you for a file name and the file path if you want it saved to a location other than the default **bin** directory. A list of all JADE methods in this schema that have not been executed is then saved to a CSV file in a table containing the **Schema**, **Type**, and **Method** columns. The type column can contain a class or a primitive type entity.

After the file has been produced, the total number of JADE methods in the schema, the number of methods included in the coverage, the number of methods not included in the coverage, and the total percentage of JADE methods executed is displayed at the left of the pane at the bottom of the Code Coverage Results Browser.

- The **Calculate Total Methods Executed Percent for Schema** command in the File menu is enabled only when a schema is selected in the **Entity** list.

When you select this command, the total percentage of JADE methods executed for this schema is calculated and displayed in the pane at the bottom of the Code Coverage Results Browser. In addition, the total number of JADE methods in the schema, the number of methods included in the coverage, and the number of methods not included in the coverage is included in the display.

The values that are displayed are cleared when you perform another file load operation.

Note This operation is manual because the calculation could take a significant amount of time for a large schema.

- The **Classes Only** command in the View menu has been renamed **Types Only**, as this covers primitive types as well as classes.
- The View menu now provides the **Display List of Entities Not Covered** command. Selecting this command toggles the display of entities that are not covered in the pane at the bottom of the Code Coverage Results Browser for classes and schemas. When you select this command from a:
 - Schema entity and the option is enabled, the list of types with JADE methods that have not been included in the coverage is displayed.
 - Class or primitive entity and the option is enabled, the list of JADE methods that have not executed is displayed.

Compact JADE

Under Compact JADE, menu accelerator keys are ignored and are not included in the displayed menu item text.

Control Changes

The following subsections contain changes to **Control** subclasses in this release.

Combo Box List Entries

The number of entries displayed in the list box portion of a combo box has been increased from a maximum of 8 entries to a maximum of 20 entries.

Disabled Text Color of Controls under Portable GUI

In JADE 6.3.04, the **DisabledTextColor** parameter in the [Jade] section of the JADE initialization file set the disabled text color of a text box under Portable GUI. This parameter now sets the disabled text color of any control under Portable GUI.

If this parameter is present and the value is not zero (**black**), all disabled controls use the specified Integer value to set the color of disabled text in a control. The default value is zero (**0**); that is, black.

This parameter is read the first time a disabled control is encountered.

Drawing on Picture Controls

You can now draw on the **Picture** control without having to handle Windows events in JADE methods. This functionality enables you to capture signatures on Windows Mobile devices running under Compact JADE. You can also use it in any JADE client environment.

The **Picture** class now provides the **startDrawingCapture** and **stopDrawingCapture** methods, which have the following signatures.

```
startDrawingCapture();  
stopDrawingCapture(): Binary;
```

Call the **startDrawingCapture** method to cause the **Picture** control to enter a special mode in which each subsequent **mouseDown** event on the control draws a dot. While the mouse is down, each **mouseMove** draws a line between the previous point and the current mouse position.

Calling this method first clears any previous drawing on the control and then sets the value of the **autoRedraw** property to **true**. All subsequent drawing activity by the user is captured and saved.

The points and lines are drawn using the color value of the **foreColor** property of the control (which is black, by default), the value of the **drawStyle** property (which is solid, by default), and the value of the **drawWidth** property (which is **1**, by default). While the control is in this special mode, any defined **mouseDown**, **mouseUp**, **click**, and **dblClick** events on the control are not called.

Call the **stopDrawingCapture** method to turn off the special drawing mode initiated by the call to the **startDrawingCapture** method and return a binary image of the client area of the control. The image size is the value of the **clientWidth** method by the value of the **clientHeight** method of the control. If the **Picture** control has no defined **picture** property value, the effective **backColor** property color is white and the **foreColor** property value is black, the image returned is a 1-bit monochrome Portable Network Graphics (PNG) image. If any of these conditions is not true, a 24-bit PNG image is returned. On Compact Pocket PC version 4.2, which does not provide image conversion, a bitmap is returned.

A **stopDrawingCapture** method call sets the value of the **autoRedraw** property to **false** and deletes all the saved drawing history. The next repaint of the control does not display the drawn image after a call to the **stopDrawingCapture** method.

If you want a different image style, instead of calling the **stopDrawCapture** method, you can perform the following equivalent calls.

```
bin := picture1.createPictureAsType(...);  
picture1.autoRedraw := false; // also turns off the special drawing  
// capture mode
```

After the special drawing mode has been turned off, any defined **mouseDown**, **mouseUp**, **click**, and **dblClick** events are again called if they occur.

If you call the **stopDrawingCapture** method and the control is not in the special drawing mode, the result is the same as a call to the **createPictureAsType** method, and no exception is raised.

Note Not all Windows Mobile devices are suitable for signature capture. Some devices do not process all of the mouse moves required for adequate signature drawing.

JadeEditMask and TextBox Classes

The **JadeEditMask** and **TextBox** controls now provide the **getTextAsInteger64** and **setTextFromInteger64** methods, which have the following signatures.

```
getTextAsInteger64(): Integer64;
setTextAsInteger64(int64: Integer64);
```

These methods behave as the existing **getTextAsInteger** and **setTextFromInteger** methods, except that they accept or return **Integer64** values.

JadeRichText::linkClicked Method

Implementing the following code in the **JadeRichText** class **linkClicked** method opens the default browser with a single click.

```
jrtResults_linkClicked(textbox: JadeRichText input;
                       link:      String) updating;
vars
  result : Integer;
begin
  result := call josShellExecute(hwnd, 'open', link, null, null, 1);
  if result <= 32 then
    app.msgBox('You do not have a default browser or the URL ' &
              link & ' could not be found', 'Error',
              MsgBox_Exclamation_Mark_Icon);
    return;
  endif;
end;
```

Looking Up a Control on a Form by Name

The **Form** class now provides the **getControlByName** method, which has the following signature.

```
getControlByName(controlName: String): Control;
```

This method enables you to look for a control on a form by name.

If the control specified in the **controlName** parameter is found on the form, the control instance is returned; otherwise null is returned.

The following example of the `getControlByName` method displays Button/16875.1 in the output viewer window.

```
vars
  frm: BingSearch;
  control: Control;
begin
  create frm;
  control := frm.getControlByName("btnSearch");
  write control;
epilog
  delete frm;
end;
```

Delta Databases

A delta database allows non-permanent (or tentative) changes to be made to a database in read-only mode. The database can be a non-SDS database, an SDS primary database, an SDS secondary database, or an RPS database.

The use case that led to the development of this feature was a requirement for applications to execute transactional updates against a read-only database; for example, an SDS secondary database. These changes are visible to all applications but are later discarded so that the SDS secondary database does not permanently diverge from the primary database of which it is a replica.

When you activate the delta database feature, a new empty database called a *delta database* is created. The original database, referred to as the *root database*, is converted to a read-only state and is said to be in *delta* mode.

When a database is in delta mode, all create, update, and delete operations on non-environmental persistent objects are re-directed to the delta database instead of being applied to the root database. In delta mode, changes to meta data objects stored in `_user*.dat` files are not permitted, which means that schema changes cannot be applied. When the root database is an SDS primary database, updates applied to the delta database are not replicated to any secondary databases.

When applications access objects created or updated in delta mode, the objects are fetched from the delta database. If objects that have been deleted in delta mode are accessed, exception 4 (*Object not found*) is raised.

The delta database is located in the `deltadb` subdirectory of the root database path.

When you deactivate delta database mode, the delta database is removed and all updates made in delta mode are discarded.

The delta database can be recovered; that is, if the server node crashes and is restarted while the delta database is active, any updates will still be present when the node is available again.

You can activate and deactivate the delta database multiple times within the lifetime of the database server node.

Getting Started

To start using delta database functionality, database server nodes must be specified to be delta database-capable. To do this, set the value of the `DeltaDatabaseCapable` command value in the [JadeServer] section of the JADE initialization file on the database server node to `true`.

JADE Initialization File Parameters

For details about the delta database parameters that can be contained in JADE initialization file sections, see the following subsections.

[JadeServer] Section

The following subsections describe the delta database parameters that can be defined in the [JadeServer] section.

DeltaDatabaseCapable Parameter

Set the **DeltaDatabaseCapable** parameter to **true** to allow the activation or deactivation of a delta database in a database server node.

The default value of **false** prevents delta database activation.

ResetDeltaModeOnRestart Parameter

Set the **ResetDeltaModeOnRestart** parameter to **true** if on restart, you want the database server node to take the root database out of delta mode and restart in normal mode (that is, to *not* use the delta database).

Caution This effectively results in the contents of the delta database being lost on server restart, as it is always recreated whenever activated.

The default value of **false** indicates that if the database server node is terminated while in delta mode, the database server node attempts to activate the delta database when it restarts.

[DeltaDb] Section

The [DeltaDb] section of the JADE initialization file is used to specify database engine initialization parameters that are used by the delta database instance. The parameters are a subset of the parameters available in the [PersistentDb] section for a standard database instance.

This discrete section is required to avoid potential conflicts when absolute paths are used for directories such as the **JournalRootDirectory**, for example. Additionally, it allows tuning parameter values to differ from those of the root database.

If this section does not exist when a new delta database instance is initialized, the section is created and parameters with their default values are written to the section.

Activating a Delta Database

Prerequisite The JADE initialization file for the database server node must have the **DeltaDatabaseCapable** parameter specified with a value of **true** in the [JadeServer] section.

You can activate a delta database dynamically from your application logic by performing one of the following actions.

- By calling the **System** class **activateDeltaDatabase** method with the **activate** parameter set to **true**
- From a command script, by executing the **ActivateDeltaDb** command in the online JADE Database Administration utility (that is, in **jdbadmin**)

Activating a delta database involves creating a new (empty) delta database and converting the current (root database) to delta mode. When the root database is in delta mode, it is in a read-only state and all object create, update, and delete operations on non-environmental persistent objects are redirected to the delta database. When an SDS native or RPS secondary database is in delta mode, database tracking is halted.

Current applications do not need to be idle when the delta database is being activated. However, database transactions that started before delta mode was activated are not allowed to commit after activation. To prevent in-progress transactions from spanning an activation boundary, activating delta mode acquires a database quiet point, which is acquired by blocking the start of new transactions in the database engine and waiting for existing transactions to complete. If a database quiet point cannot be achieved within the number of seconds specified by the **MaxWaitForQuietPoint** parameter in the [PersistentDb] section of the JADE initialization file, activation fails and exception 3077 (*Maximum time to wait for quiet point was exceeded*) is raised.

Note **MaxWaitForQuietPoint** is an existing database parameter.

Deactivating a Delta Database

You can deactivate a delta database dynamically from your application logic by performing one of the following actions.

- By calling the **System** class **activateDeltaDatabase** method with the **activate** parameter set to **false**
- From a command script, by executing the **DeactivateDeltaDb** command in the online JADE Database Administration utility (that is, in **jdbadmin**)

Deactivating a delta database involves taking the root database out of delta mode and removing the delta database.

The process that requests the deactivation must not have any persistent objects locked, must not be in transaction state, and must not be executing any methods that have a persistent user object as the receiver. If these conditions are not satisfied, deactivation is abandoned and system exception 1163 (*Could not change delta database mode because not all processes are idle*) is raised.

Before the delta database can be deactivated, all current applications (apart from the application making the request) must be idle, not in transaction state, and have no persistent locks held. To be idle, an application must not be currently executing any methods invoked via the JADE object manager; that is, its object manager call stack should be empty. The deactivation attempt is immediately abandoned if there are any idle processes that have persistent objects locked or that are in transaction state.

When a deactivation request is received, each application is allowed to proceed normally until its JADE call stack is empty. When a process is idle, any further attempt to invoke a method is held up until deactivation is completed or abandoned. The process making the request and JADE tools such as the JADE Monitor and the SDS Administration application are exempt from this.

Any attempts to start up new applications are also stalled until deactivation is completed or abandoned.

If the request cannot be completed because not all applications have become idle within the time specified by the **timeout** parameter, the deactivation attempt is abandoned and system exception 1163 (*Could not change delta database mode because not all processes are idle*) is raised.

SDS and RPS Operational Characteristics

This section describes delta database operational characteristics of SDS and RPS.

Database Tracking and RPS Replication

When an SDS secondary database enters delta mode, database tracking is stopped. In an RPS node, the **Datapump** application is not stopped but remains in an idle state.

When a secondary database exits delta mode, database tracking is restarted (if it is not disabled) and on an RPS node, the **Datapump** application is restarted if it was stopped for some reason and the **AutoStartDataPump** parameter in the [JadeRps] section of the JADE initialization file is set to **true**.

Administrative Restrictions

When an SDS secondary or primary database is in delta mode, specific administrative operations invoked by calling SDS-related methods in the **JadeDatabaseAdmin** class are not permitted; attempts to execute a disallowed operation that is processed synchronously fail and an exception is raised.

You cannot execute the following SDS secondary administrative operations from a secondary database that is in delta mode.

JadeDatabaseAdmin Class Method	SDS Administration Command
sdsStartTracking	Start Tracking
sdsResume	Resume
sdsReplayNextJournal	Replay Next Journal
sdsInitiateHostileTakeover	Initiate Hostile takeover

You cannot execute the following SDS secondary administrative operations from a primary database server when the selected secondary database is in delta mode.

JadeDatabaseAdmin Class Method	SDS Administration Command
sdsStartTrackingAt	Start Tracking (at selected secondary)
sdsResumeAt	Resume (at selected secondary)
sdsReplayNextJournalAt	Replay Next Journal (at selected secondary)
sdsInitiateTakeover	Initiate Takeover (to selected secondary)

You cannot execute the following SDS primary administrative operation from a primary database server when the primary database is in delta mode.

JadeDatabaseAdmin Class Method	SDS Administration Command
sdsInitiateTakeover	Initiate Takeover (to any secondary)

New SDS_ReasonDeltaModeEntered Tracking Stopped Reason Code

When database tracking is stopped because the database entered delta mode, the reason code passed in the **userInfo** parameter to the **SDS_TrackingStopped** (17388) system event is defined by the **SDS_ReasonDeltaModeEntered** (which has an Integer value of 12) global constant in the **SDSStopTrackingCodes** category.

Caveats that Apply when Using Class Extent Methods in Delta Mode

Methods defined in the **Class** class that operate on the class extent (for example, **firstInstance** and **lastInstance**), *all instance* variants, and the **Class** class **instances** property virtual collection are executed on both the root and delta database but the merged result set may not be the same as the result set obtained outside of delta mode.

The differences in behavior when using class extent methods in delta mode are as follows.

- Any class extent method can return a reference to an object deleted in delta mode; for example, if the first instance of a class is deleted in delta mode, a reference to this object is still returned by the **Class** class **firstInstance** method.
- The **Class** class **countPersistentInstances** method and calls to **Class.instances.size** include instances that were deleted in delta mode.

New Methods Defined in the System Class

The following subsections describe new **System** class methods in this release.

activateDeltaDatabase Method

The **System** class **activateDeltaDatabase** method has the following signature.

```
activateDeltaDatabase(activate: Boolean;  
                     timeout: Integer): Boolean;
```

Call the **activateDeltaDatabase** method to perform one of the following actions.

- Create a delta database and activate delta mode.
- Deactivate delta mode and delete the delta database.

Use the **activate** parameter to specify whether the delta database is being activated (**true**) or deactivated (**false**).

Use the **timeout** parameter to specify a maximum number of seconds to allow for deactivation to take place. A default timeout of 60 seconds is used if the specified value is zero (**0**). The **timeout** parameter ignored when the **activate** parameter is set to **true**.

You cannot deactivate the delta database until all current processes apart from JADE tools (for example, the JADE Monitor and the SDS Administration application) and the process making the request are idle. The deactivation attempt is abandoned if this does not happen within the time specified in the **timeout** parameter, the delta database remains activated, and system exception 1163 is raised.

The return value indicates the delta database status prior to the method call. If the delta database is active at the time of the call, **true** is returned, or if it was inactive, **false** is returned.

The following code fragment is an example of activating a delta database.

```
system.activateDeltaDatabase(true, 0);
```

The following code fragment is an example of deactivating a delta database.

```
//Deactivate the Delta Database. Allow up to 2 minutes for deactivation.  
system.activateDeltaDatabase(false, 120);
```

The following system exceptions can be raised from an **activateDeltaDatabase** method call.

- 1162 - *The system is not delta database capable*
Exception 1162 is raised if the database server node is not specified to be delta database-capable.
- 1163 - *Could not change delta database mode because not all processes are idle*
Exception 1163 is raised and the deactivation attempt is abandoned if a deactivation request cannot be completed because not all applications have become idle within the time specified in the timeout parameter.
- 1165 - *A delta database transition request is already in progress*
Exception 1165 is raised if a deactivation or activation request is currently in progress.

getDeltaDatabaseStatus Method

The **System** class **getDeltaDatabaseStatus** method has the following signature.

```
getDeltaDatabaseStatus(): Integer;
```

The **getDeltaDatabaseStatus** method returns the delta database status.

The return values that indicate the current delta database status are listed in the following table.

Integer Value	Descriptio
0	The server node is not specified to be delta database-capable
1	Inactive
2	Active
3	Being activated (reserved for future use)
4	Being deactivated

Restarting a Delta Database

If a database server node is shut down or terminates abnormally while the delta database is active with the **ResetDeltaModeOnRestart** parameter set to **false**, the node restarts with the delta database active and the delta database is recovered.

When the database server node is started with the delta database active, a check is made to ensure that the delta database matches the root database; that is, it is the delta database that was created when the root database most recently went into delta mode. If it does not match, the database server node will not start and error *Delta database ID mismatch* will be logged in the **jommsg.log** file.

If a valid version of the delta database cannot be restored, you can restart a database server node in non-delta mode by specifying the **ResetDeltaModeOnRestart** parameter with a value of **true** in the [JadeServer] section of the JADE initialization file. When restarted, the database server node takes the root database out of delta mode and then restarts it in normal mode; that is, not using a delta database.

Caution This results in the contents of the delta database being lost, as it will be re-created when the delta database is next activated.

If all else fails, you can use the **clearDeltaMode** command in the batch JADE Database utility (**jdbutilb**) to clear the delta mode state persisted in the root database control file.

JADE Database Batch Utility (jdbutilb) Command

The following subsection contains the new batch JADE Database utility command in this release. (See also “[Batch Database Utility](#)”, earlier in this document.)

clearDeltaMode Command

The **clearDeltaMode** command enables you to clear the delta mode state preserved in the database control file. This may be necessary if, for any reason, it is not possible to deactivate delta mode online.

The syntax of the **clearDeltaMode** command is as follows.

```
jdbutilb path=database-path ini=initialization-file-name clearDeltaMode
```

Clearing delta mode state removes any delta mode restrictions; for example:

```
jdbutilb path=d:\salesdb ini=d:\salesdb\jade.ini clearDeltaMode
```

JADE Database Administration Utility (jdbadmin) Commands

The following subsections contain the new batch JADE Database Administration utility commands in this release.

ActivateDeltaDb Command

The **ActivateDeltaDb** command creates a new delta database in the **deltadb** subdirectory of the root database and converts the root database to delta mode.

The syntax of the **ActivateDeltaDb** command is as follows.

```
jdbadmin action=ActivateDeltaDb
        [path=database-path]
        [ini=initialization-file-name]
        [server=multiUser]
```

The following is an example of the **ActivateDeltaDb** command.

```
jdbadmin path=d:\salesdb ini=d:\salesdb\jade.ini action=ActivateDeltaDb
```

DeactivateDeltaDb Command

The **DeactivateDeltaDb** command deactivates delta mode and removes the associated delta database.

The syntax of the **DeactivateDeltaDb** command is as follows.

```
jdbadmin action=DeactivateDeltaDb
        [path=database-path]
        [ini=initialization-file-name]
        [server=multiUser]
        [timeout=integer-value]
```

The optional **timeout** parameter specifies a maximum number of seconds to allow for deactivation to take place.

The delta database cannot be deactivated until all current processes apart from JADE tools (for example, the JADE Monitor and the SDS Administration application) and the process making the request are idle with no persistent locks and not in transaction state. The deactivation attempt is abandoned if this does not happen within the specified time, and the delta database remains activated. A default timeout of 60 seconds is used if you specify zero (**0**) in the **timeout** parameter.

The following is an example of the **DeactivateDeltaDb** command.

```
jdbadmin path=d:\salesdb ini=d:\salesdb\jade.ini
action=DeactivateDeltaDb timeout=300
```

Development Environment Change

This section contains the JADE development environment change in this release.

Splitting the Editor Pane View

You can now split JADE development environment windows that display JADE method source (that is, the editor pane) into two horizontal views of the same method.

You can scroll these views independently, to view and edit different parts of the active document at the same time. Any changes that you make in one view are reflected in the other.

➤ To open or manipulate the second view, perform one of the following actions.

- Position the mouse cursor on the split bar just above the status line so that it changes to the resize pointer and then click and drag the split bar to where you want it positioned.
- Use the SHIFT+CTRL+W shortcut keys to open or close (that is, toggle) the lower view.
- Use the CTRL+W shortcut keys to open the lower view if it is not already open or to switch focus between the two views if the lower view is already open.

In addition to the editor key binding actions documented under “Maintaining Editor Key Bindings” in Chapter 2 of the *JADE Development Environment User’s Guide*, you can now perform the following key binding actions.

Action	Description
SplitOpenClose *	Toggles (opens or closes) the lower view of an editor pane
SplitOpenRefocus *	Opens the lower view of an editor pane or toggles focus between the two views

The asterisk character (*) indicates that these actions cannot be bound to another keystroke, but you can disable the default binding by selecting the **NoAction** action.

Encoding and Decoding the Base64 Format

The **Binary** and **String** primitive types now provide the methods documented in the following subsections.

Binary::**base64Encode** Method

The **Binary** primitive type **base64Encode** method encodes the binary of the receiver into an ASCII string, using the base64 encoding technique defined in RFC 1521.

Base64 encoding:

- Allows 8-bit data to be converted, so that it can be transmitted over a protocol that allows only 7-bit characters.
- Gives enhanced privacy if the source data is standard ASCII text, as the message is no longer in clear text when it is transmitted.

The returned string is represented in lines of no more than 76 characters each and it is terminated with a carriage return/line feed (CR/LF) character sequence.

Binary::*base64EncodeNoCrLf* Method

The **Binary** primitive type **base64EncodeNoCrLf** method encodes the binary of the receiver into an ASCII string, using the base64 encoding technique defined in RFC 1521. Base64 encoding:

- Allows 8-bit data to be converted, so that it can be transmitted over a protocol that allows only 7-bit characters.
- Gives enhanced privacy if the source data is standard ASCII text, as the message is no longer in clear text when it is transmitted.

The returned string is an ASCII string resulting from the encoding of the receiver using the Base64 encoding technique defined in RFC 1521. Unlike the **base64Encode** method, the output is not broken up into lines; that is, it does not contain carriage return and line feed (**Cr** and **Lf**) characters.

String::*base64Decode* Method

The **String** primitive type **base64Decode** method takes a base64-encoded string and decodes it into a binary. A base64-encoded message contains characters from the following alphabet.

```
ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/
```

The message can also contain line breaks (the CR/LF character sequence) inserted by the base64 encoding algorithm. These line breaks are ignored by the decoder, as are any characters that are not in the base64 alphabet.

Base64 Encoding and Decoding Example

The following method is an example of the base64 encoding and decoding methods.

```
vars
  bin: Binary;
  file: File;
begin
  create file;
  file.fileName := 'd:\temp\harry.jpg';
  file.kind := File.Kind_Binary;
  file.open;
  bin := file.readBinary(file.fileLength);
  write 'original length = ' & bin.length.String;
  write 'base64Encode length = ' & bin.base64Encode().length.String;
  write 'base64EncodeNoCrLf length = ' &
    bin.base64EncodeNoCrLf().length.String ;
  write 'base64Decode length = ' &
    bin.base64Encode().base64Decode().length.String;
  write 'base64Decode length (from NoCrLf) = ' &
    bin.base64EncodeNoCrLf().base64Decode.length.String;
  file.close;
epilog
  delete file;
end;
```

The method in this example outputs the following to the Jade Interpreter Output Viewer window.

```
original length = 5062
base64Encode length = 6928
base64EncodeNoCrLf length = 6752
base64Decode length = 5062
base64Decode length (from NoCrLf) = 5062
```

Note The length of the encoded string is about a third longer. Decoding results are the same, regardless of whether the string was encoded with the carriage return/line feed (CR/LF) character sequence or not.

Error Messages

This section describes new and changed error messages in this release. (See also “[New Error Messages](#)”, under “[Batch JADE Database Utility](#)”, and “[System Sequence Number Errors](#)”, elsewhere in this document.)

1040 - Failed to open schema agent

The existing error 1040 can now also be returned as an error result when attempting to sign on to an application but the schema is not available; for example, because of a reorganization.

If this error occurs when signing on, wait for whatever it is that made it unavailable to complete (for example, a reorganization) and then try signing on again.

1160 - Background Process request was not completed

Error 1160 occurs when the JADE background process for a node is unable to complete a request; for example, signing a process on or off.

If this error occurs, retry the operation that caused it to be raised. If the problem persists, refer to your JADE messages log file (**jommsg.log**) for error details.

1161 - Background Process request encountered a lock error

Error 1161 occurs if a lock-related error occurred while the JADE background process was carrying out a request; for example, signing a process on or off. Further details of the error is written to your JADE messages log file (**jommsg.log**).

If this error occurs, retry the operation that caused it to be raised. If the problem persists, refer to your JADE messages log file (**jommsg.log**) for error details.

1265 - Environmental object operation is out of scope for process

The existing exception 1265 now occurs for the following **Process** class methods if the receiver is not the current process.

- `isInTransactionState`
- `isInTransientTransactionState`
- `isInLoadState`
- `isInLockState`

- `isInExceptionState`

Although the result returned by a call to these methods always related to the current process only in earlier releases, no exception was raised if a different process was the receiver.

3219 - Cannot run SDS secondary server in SingleUser

Error 3219 occurs when an attempt is made to start an SDS secondary server node in single user mode; that is, when the **server** command line parameter is set to **localServer** and the value of the **DatabaseRole** parameter is specified as **SecondaryRole** in the [SyncDbService] section of the JADE initialization file.

This complements the existing 3206 error (that is, *Operation not permitted on a secondary database*) that is displayed when a database operation that is considered invalid on a secondary database is attempted.

If an SDS primary server node is started in single user mode, full SDS functionality is not available (although updates are still tracked).

6429 - Cannot mark the default map as partitionable

Error 6429 occurs when you attempt to load a schema with the default map file marked as partitionable.

Fault Handling

This section contains the fault handling changes in this release.

TCP/IP Disconnect Logging Suppression

You can now suppress the logging of TCP/IP normal disconnections, by setting the value of the new **LogTcpNormalDisconnects** parameter in the [FaultHandling] section of the JADE initialization file to zero (0).

The default value of **1** retains the behavior of earlier JADE 6.3 releases; that is, normal TCP/IP disconnections are logged.

Thread Execution and Callback Logic Exception Handling

When thread execution and callback logic detect exceptions and invoke unhandled exception processing, if the [FaultHandling] section of the JADE initialization file contains the **Disabled** parameter with a value of **true**, structured exceptions are passed on to the system default handler for attention; for example, to Windows error reporting or to the JADE Just-In-Time (JIT) debugger.

This parameter is not written to the [FaultHandling] section of the initialization file by default, so the implied value is **false**, which means that you must define this parameter with a value of **true** if you want it read when an error occurs.

HTML Documents

This section contains the HTML document changes in this release.

Finding and Updating HTML Document Source

The **Schema** class `getHtmlDocumentSource` and `setHtmlDocumentSource` methods now use the receiver schema to find and update the HTML document source, instead of the current schema.

Translating Strings in the HTML Document Source

You can now use the `<JADE_TAG>` to define translatable strings in the HTML document source. The `JADE_TAG` uses the new `_translate` attribute to do this, as shown in the following example.

```
<JADE_TAG name=nameLabel _translate=translatableStringName>
```

When the HTML is generated for a document containing the `_translate` attribute, a lookup of the *translatableStringName* string is performed, using the locale defined for the session.

If the locale does not exist, it is set to null (""), or the string does not exist in the list of translatable strings, the `<JADE_TAG>` tag is replaced by the *translatableStringName* string itself. If the string does exist, the `<JADE_TAG>` tag is replaced by the value of the translatable string for that locale.

A selective extract of a **JadeHTMLClass** subclass extracts the translatable string definitions of all translatable strings that are defined in the HTML document.

Note As a patch extract extracts only what is defined in the patch, if a translatable string defined in an HTML document is not defined in a patch, it is not extracted with that patch.

JadeAuditAccess Class

In earlier releases, the **JadeAuditAccess** class `loadDescription` method used the path of the journal being processed by default if the value of the `descriptionPath` property was null ("").

This is unhelpful when processing is in-band (that is, the system is analyzing activity that has occurred within the same system), as this default location is not the location into which description files are generated. Description files are generated into the journal root directory by default or set by specifying the required directory in the **JournalRootDirectory** parameter in the [PersistentDb] section of the JADE initialization file.

When processing is out-of-band (that is, when the system is analyzing activity that has occurred within another system), it is not appropriate to use current system parameters for default locations. There is no requirement for journal directory hierarchies to be the same or for a hierarchy to exist at all. In this case, it is feasible to have a `descriptionPath` property value of null default to the path of the journal that is being processed.

To accommodate both in-band and out-of-band situations, the `loadDescription` method now defaults a `descriptionPath` property value of null as follows.

- When the path of the journal being processed has *current* as the final node, the description path becomes the journal path without that final node. This is appropriate when processing in-band, as the default location of a call to the `loadDescription` method will be the journal root directory and it will match the description generate location.

When processing out-of-band, the description path is the journal path.

For example, when processing is:

- In-band, the journal path is `x:\prod\system\journals\current\Examples` and the value of the `descriptionPath` property defaults to `x:\prod\system\journals\`

- Out-of-band:
 - The journal path is `x:\track\incident\1432\` and the value of the **descriptionPath** property defaults to `x:\track\incident\1432\`
 - The journal path is `x:\investigations\current\` and the value of the **descriptionPath** property defaults to `x:\investigations`

Additionally, a call to the **loadDescription** method no longer clears the values of the **descriptionFilename** and **descriptionPath** properties when the description file cannot be opened. These properties now always contain the values determined the last time a description file load was attempted in the current session.

JadeBytes Class Changes

The following changes have been made to the handling of the **JadeBytes** instances in this release.

- The **JadeBytes::setContent** method now allocates a best fit tail segment from the range of valid segment sizes.
- When the **JadeBytes::setContent** method is used to replace the binary contents of an existing **JadeBytes** object and the content length can be accommodated in fewer segments, smaller tail segments, or both fewer segments and small tail segments, the container is truncated.
- The **JadeBytes::getStatistics** method has been enhanced to include the physical size of the tail segment and its content length.
- The **JadeBytes::truncate** method now reduces the size of the tail segment to the best fit size when called with the length equal to its current content length.

JadeHttp Configuration

If you want to purge all files in the virtual directory that are not read-only, you can now specify the:

- **PurgeDirectoryRule** parameter with a value of **AllWritable** in the [*application-name*] section of the Internet Information Server (IIS) **jadehttp.ini** file.
- **PurgeDirectoryRule** directive with a value of **AllWritable** within the **<Location>** directive of the Apache HTTP Server configuration **mod_jadehttp** module.

The default value of **Default** for this parameter removes any standard files of type **.jpg**, **.png**, or **.gif** that are more than 12 hours old.

Merge Iterator Restrictions Removal

The following merge iterator restrictions have now been removed.

- The **MergeIterator** class **addCollection** method no longer requires that all collections have the same keys.
Iteration is now supported over collections that have a common compatible subset of keys. Compatible keys have the same type, length, precision, and scale factor (where applicable). The key attributes must also be the same; that is, ascending, duplicates, and sort order.
- There is no longer the requirement that the membership of collections being iterated is the same.
You can now add collections with completely unrelated classes, as long as at least the first key is the same.

An exception is raised if the **MergeIterator** class **startAtObject** method is called and the membership of all collections being iterated is unrelated.

ODBC Changes

This section contains the ODBC changes in this release.

ODBC Driver

The ODBC driver now handles large result sets by writing the output to temporary files, if required. This will happen automatically without user intervention. Some parameters in the [JadeOdbc] section of the JADE initialization file can be set to control file location and memory usage.

In the [JadeOdbc] section of the JADE initialization file for the ODBC service application or the standard client, set the following parameters to control the result set handling.

- **ServerResultSetBufferSize**

The **ServerResultSetBufferSize** parameter, which accepts the metric format, specifies the size of memory to be allocated for the result set buffer. The default value is **1M**. If the result set exceeds the buffer size, the result set is written out to a temporary file. A buffer is allocated for each query request.

The parameter is read when the connection is established.

- **ExtractDirectory**

The **ExtractDirectory** parameter specifies the location where the extract files for the result set are to be written. The default value is the **temp** directory.

The parameter is read when the extract file is required to be written.

- **SortMemory**

The **SortMemory** parameter specifies the size of memory to be allocated when sorting the result set. This value is used when result sets that overflow the result set buffer size must be sorted (for the **ORDER BY** or **DISTINCT** clause).

Multiple allocations of the specified **SortMemory** size can occur, depending on the result set size. The default value is the value of the **ServerResultsSetBufferSize** parameter.

The value is read when a result set which was written to an extract file is sorted.

The **MaximumResultSetSize** parameter is obsolete and is ignored.

Thin Client Data Source Name (DSN)

The JADE ODBC Thin Client Setup dialog now provides the **Client Buffer Size** combo box, which enables you to set the size of the client buffer. Valid values displayed in the drop-down list are **1M** (the default), **2M**, **5M**, and **10M**.

The size of the thin client buffer determines the maximum data transfer size for the result set from the ODBC service application to the presentation client. The actual maximum size is the minimum of the selected client buffer size and the value of the **ServerResultSetBufferSize** parameter.

ODBC Execution Trace Output

The [JadeOdbc] section of the JADE initialization file can now contain the **QueryExecutionTraceOn** parameter, which controls the output of query execution tracing.

Set this parameter to **true** if you want to output query execution trace information to the **jommsg** log file during the execution of the query. (By default, the value of this parameter is **false**.)

This parameter is read when the ODBC connection is established.

Process::isRunningScript Method

The **Process** class now provides the **isRunningScript** method, which has the following signature.

```
isRunningScript(): Boolean;
```

Use this method to determine whether the process is running Workspace code or a **JadeScript** method.

Note As a network message is sent to the node on which the process is running, you should take this overhead into account when using the **isRunningScript** method.

Relational Population Service (RPS)

This section contains the RPS changes in this release. (See also “[Delta Databases](#)”, earlier in this document.)

Automatically Restarting the Datapump Application

If the **Datapump** application fails with a connection or timeout error, the [JadeRps] section of the JADE initialization file now provides the following parameters.

```
[JadeRps]
AutoRestartOnError=true
AutoRestartRetryLimit=10
AutoRestartDelay=10
```

The **AutoRestartOnError** parameter specifies whether or not the **Datapump** application will be automatically restarted after a connection or timeout error. The default value is **false**.

If the **AutoRestartOnError** parameter is set to **true**, the **AutoRestartRetryLimit** parameter specifies the number of retries (the default value is 10) and the **AutoRestartDelay** parameter specifies the number of seconds to wait between retries (the default value is 10).

The following ODBC error states define connection or timeout errors that trigger the automatic restart (if the value of the **AutoRestartOnError** parameter is **true**).

ODBC Error Number	Description
08001	Unable to establish connection
08004	Server rejected the connection
08007	Connection failure during transaction
08S01	Communication link failure
HYT00	Timeout expired
HYT01	Connection timeout expired.

Fault Handling when the Datapump Application Terminates Abnormally

The [JadeRps] section of the JADE initialization file can now contain the **DatapumpErrorFileDirectory** parameter, which specifies the location of errors file created when the **Datapump** application on an RPS node terminates abnormally.

The *location* value is the location in which the **.err** files will be created. If this parameter does not exist in the initialization file, no **.err** files are created. The *location* value follows the same rules as those of the **LogDirectory** parameter in the [JadeLog] section of the JADE initialization file.

If the **Datapump** application fails with an error (that is, connection errors, table mismatch errors, row creation, update, or delete errors, table inconsistency, extract or load errors, or other exceptions), the file *location/rps_unique-id.err* is created.

The file contains available information about the error, including any ODBC errors reported, the exception stack or both ODBC errors and the exception stack.

Running a User Data Pump Extract on an SDS Secondary or RPS Node

The new **RelationalView::extractDataAll** and **JadeDatabaseAdmin::rpsExtractDataAll** methods extract all tables in the RPS mapping, using the specified parameter values.

In addition, the **RPSAdminWS** application provided by JADE in the **JadeMonitorSchema** schema (and documented under “Calling the RPS Manager from a Web Service Consumer”, in Chapter 2 of the *JADE Synchronized Database Service (SDS) Administration Guide*) now exposes the **extractDataAll** method.

Note You can execute the **extractDataAll** method on an SDS secondary *or* an RPS node, but not on the primary node. You can execute the **rpsExtractDataAll** method on an RPS node only, not on the primary node. Running an RPS extract on an SDS node causes tracking to be stopped during the extract.

If you extract the data via the RPS Manager on the SDS node, you must specify the user **Datapump** application and schema or the **<default>** value in the **DataPumpApplication** parameter in the [JadeRps] section of the JADE initialization file.

The **extractDataAll** or **rpsExtractDataAll** method, which has the following signature, extracts all tables in the RPS mapping, using the specified parameter values.

```
extractDataAll | rpsExtractDataAll (executionLocation:      Integer;
                                   scriptFilePath:         String;
                                   dataFilesPath:          String;
                                   rdbDataFilesPath:       String;
                                   rdbName:                String;
                                   extractHistoricalTables: Boolean;
                                   serverName:             String;
                                   extractWorkers:        Integer;
                                   extractOrder:          Integer;
                                   extractFirst:           String;
                                   userDataPumpSchema:     String;
                                   userDataPumpApp:        String);
                                   Process;
```

See the **RelationalView::extractData** or **JadeDatabaseAdmin::rpsExtractData** method for details about the parameters **executionLocation** through **extractWorkers**.

The **extractOrder** parameter specifies the order in which the tables are to be extracted, using the following new constants defined in the **RelationalView** or **JadeDatabaseAdmin** class.

- **ExtractOrderDefault** (0), indicating that no extract order is specified.
- **ExtractOrderClassInstances** (1), which outputs tables in order of the number of instances of the class, from highest to lowest.

Note Determining the number of instances may delay the start of extraction.

- **ExtractOrderSelectedFirst** (2), which first outputs tables specified in the **extractFirst** parameter and then uses the default order. When specifying values for the **extractFirst** parameter, table names are delimited by semicolons.

For details about worker allocation extraction order, see “Selecting Tables” under “Extracting Data from the JADE Database”, in Chapter 2 of the *JADE Synchronized Database Service (SDS) Administration Guide*.

Report Writer

The [JadeReportWriter] section of the JADE initialization file can now contain the **QueryOptimizationTraceOn** parameter, which switches on or off the Query Optimization logging diagnostics.

Set this parameter to **true** if you want to output optimization trace information to the **jommsg** log file during the execution of the query. (By default, the value of this parameter is **false**.)

Schema::extractControlIdsCSV and extractControlIdsCSVforSchema Methods

The **Schema** class now provides the **extractControlIdsCSV** and **extractControlIdsCSVforSchema** methods, which enable you to write a Comma-Separated Values (CSV) file containing the generated Window control identifiers used by JADE when controls are created under Microsoft Windows. These control identifiers are used by testing tools to identify the required elements on a form. (JADE does not use them itself).

The identifier for a control on a form created in the JADE Painter is retained for the lifetime of the form (unless the control is deleted and re-added via the JADE Painter).

The **Schema** class **extractControlIdsCSV** and **extractControlIdsCSVforSchema** methods have the following signatures.

```
extractControlIdsCSV();
extractControlIdsCSVforSchema(file: File);
```

The **extractControlIdsCSV** method writes a CSV file containing an entry for every control on every form in the current schema and all of its subschemas. This method displays a common dialog, which enables you to identify the directory and name of the output CSV file (which defaults to **controlIds.csv**). You can call this method, for example, from a **JadeScript** method defined in the required schema.

The source of the `extractControlIdsCSV` method is as follows, if you want to vary the calling approach.

```

extractControlIdsCSV();
vars
  file : File;
  cmd  : CMDFileSave;
  subs : SchemaNDict;
  scm  : Schema;
begin
  create cmd;
  cmd.fileName := "controlIds.csv";
  if cmd.open = 0 then
    create file transient;
    file.openOutput(cmd.fileName);
    create subs transient;
    subs.add(currentSchema);
    while subs.size() > 0 do
      scm := subs.first;
      scm.extractControlIdsCSVforSchema(file);
      subs.remove(scm);
      scm.getSubschemas(subs);
    endwhile;
    file.close;
  endif;
epilog
  delete cmd;
  delete file;
  delete subs;
end;

```

The `extractControlIdsCSVforSchema` method writes a CSV file containing an entry for every control on every form in the receiver schema only. This method assumes that the file specified in the `file` parameter is already open.

Synchronized Database Service (SDS) Tracking

The `JadeDatabaseAdmin` class now provides the `getReasonTrackingStopped` method, which has the following signature.

```
getReasonTrackingStopped(reason: Integer): String;
```

This method returns a string containing a textual description (for example, "**Transition Halt**") of the `SDSStopTrackingCodes` global constant reason code passed in the `userInfo` parameter of the system `SDS_TrackingStopped` event.

System Sequence Numbers

System sequence numbers enable applications in your JADE environment to have a series of consecutive unique numbers without requiring the use of object locks to ensure that the numbers are unique. For details about creating system sequence numbers and obtaining the next system sequence number, see the following subsections.

Each environment can have an unlimited number of system sequence numbers, each of which is identified by a unique user-assigned name of up to 60 characters.

System::getSystemSequenceNumberNext Method

The **System** class now provides the **getSystemSequenceNumberNext** method, which has the following signature.

```
getSystemSequenceNumberNext(name: String): Integer64;
```

This method increments the current value of the system sequence number specified in the **name** parameter and returns the new value. The returned value is in the range 1 through **Max_Integer64**.

If the specified system sequence number has not yet been created by a call to the **createSystemSequenceNumber** method, the **getSystemSequenceNumberNext** method returns zero (**0**).

When a call to the **getSystemSequenceNumberNext** method has returned **Max_Integer64** for the specified system sequence number, all subsequent calls for that system sequence number raise exception 1456.

All access to the system sequence number table is single-threaded independently of process transaction state.

It is your responsibility to ensure that the initial value passed to the **createSystemSequenceNumber** method will not cause a call to the **getSystemSequenceNumberNext** method to return a number that has already been used. The application must determine the highest number in a sequence that it has assigned to an object stored in the database whenever the **getSystemSequenceNumberNext** method returns zero, advising the application to invoke the **createSystemSequenceNumber** method.

If an application is granted a number in a specified sequence but does not store it persistently because the transaction is aborted, there will be a gap in the stored number sequence.

A system sequence number could be an invoice number, student number, or transaction number, for example.

In the method in the following example, the next available customer number is returned, where the customer number is used as a key in a member key dictionary owned by a **Company** object.

```
getNextCustomerNumber(): Integer;
constants
    SSN_CustomerNumber :String = "MySchema::CustomerNumber";
vars
    nextNumber : Integer64;
    coy : Company;
    cust : Customer;
begin
    nextNumber := system.getSystemSequenceNumberNext(SSN_CustomerNumber);
    if nextNumber = 0 then
        coy := Company.firstInstance();
        if coy <> null then
            cust := coy.allCustomersByNumber.last();
            if cust <> null then
                nextNumber := cust.number;
            endif;
        endif;
        system.createSystemSequenceNumber(SSN_CustomerNumber, nextNumber);
        nextNumber :=
            system.getSystemSequenceNumberNext(SSN_CustomerNumber);
    endif;
    return nextNumber.Integer;
end;
```

System::`createSystemSequenceNumber` Method

The **System** class now provides the **`createSystemSequenceNumber`** method, which has the following signature.

```
createSystemSequenceNumber(name: String;
                           initialValue: Integer64);
```

This method creates a system sequence number with the name specified in the **name** parameter and assigns the value specified in the **initialValue** parameter to its current value.

If a system sequence number with the specified name already exists, no error is reported and the current value is left unchanged.

All access to the system sequence number table is single-threaded independently of process transaction state.

If you do not specify a name, the specified name is longer than 60 characters, or it contains embedded null values, exception 1454 is raised.

If the specified initial value is less than zero, exception 1455 is raised.

There is no restriction to the number of system sequence numbers you can create.

It is your responsibility to ensure that the initial value passed to the **`createSystemSequenceNumber`** method will not cause a call to the **`getSystemSequenceNumberNext`** method to return a number that has already been used. The application must examine the database to determine the highest number that is in use in a specific sequence and pass that value to the **`createSystemSequenceNumber`** method.

For an example of the use of this method, see the **`getSystemSequenceNumberNext`** method in the previous section.

System Sequence Number Errors

This section describes the system sequence number errors that can be raised.

1454 – The `SystemSequenceNumber` name is invalid

Error 1454 occurs if the specified system sequence number is null, it is longer than 60 characters, or it contains embedded null values.

1455 – The `SystemSequenceNumber` initial value cannot be negative

Error 1455 occurs if the initial value passed to the **`createSystemSequenceNumber`** method is not greater than or equal to zero (0).

1456 – The `SystemSequenceNumber` has reached the maximum value (`Max_Integer64`)

Error 1456 occurs if the maximum value has already been returned by a call to the **`getSystemSequenceNumberNext`** method for the specified system sequence number.

Update Locks

An update lock can now be performed on a non-collection object. For an update lock to be held, the process must be in transaction state.

If you call `process.useUpdateLocks(true)`, implicit update locks are acquired when an object (a collection or a non-collection) is updated.

Updating a Partitioned Database File

In earlier releases, the source file was locked to prevent concurrent updates and if an application attempted to update a file that was being partitioned, the update was rejected with a system exception (error 3116 - Database file is locked for reorganization). For details about database file partitioning, see Chapter 2 of the *JADE Database Administration Guide*.

The online batch JADE Database Administration utility (the `jdbadmin` program) **MakePartitioned** command now provides an **updatesAllowed** parameter, which allows an administrative user to partition a file while allowing applications to continue updating the file being partitioned. If the parameter is not specified, the value defaults to **false**.

If you execute the **MakePartitioned** command with **updatesAllowed=true**, when the file conversion process has completed, updates audited in transaction journals are applied to the output file. This recovery phase is referred to as *file synchronization*. During the initial file recovery or synchronization phase, further updates to the file are permitted. However, when the file is ultimately instantiated, a database quiet point is acquired. This quiet point will momentarily block active transactions from committing while remaining updates are applied and the partitioned file is instantiated.

The syntax of the **MakePartitioned** command is now as follows.

```
jdbadmin action=MakePartitioned
         [path=database-path]
         [ini=initialization-file-name]
         [server=multiUser|singleUser]
         file=file-name
         [method=method-name]
         [modulus=integer-value]
         [updatesAllowed=true|false]
```

The following is an example of the **MakePartitioned** command.

```
jdbadmin path=c:\JADE_JRF301 ini=c:\JADE_JRF301\jade.ini
         action=MakePartitioned file=sales modulus=36 updatesAllowed=true
```

Web Services

This section contains the Web service changes in this release.

Creating and Deleting Virtual Directory Files

The methods documented in the following subsections enable you to create and delete virtual directory files in Web service applications. (In earlier releases, you could do this only for Web-enabled forms for JADE-generated images.)

createVirtualDirectoryFile Method

Signature `createVirtualDirectory(fileName: String;
 fileContents: Binary;
 retain: Boolean): Integer;`

The `JadeWebServiceProvider` class `createVirtualDirectoryFile` method passes files to the `jadehttp` library, which creates the specified file in the directory specified by the value of the **VirtualDirectory** parameter in the `jadehttp.ini` file.

The method parameters are listed in the following table.

Parameter	Description
fileName	Name of the file to be created in the virtual directory
fileContents	Binary holding the file contents
retain	Creates read-only files when set to true or standard files when set to false

This method returns zero (**0**) if the file creation is successful or it returns a non-zero error code if the create fails.

Note You must call this method during the processing cycle of the message.

***isVDFilePresent* Method**

Signature `isVDFilePresent(fileName: String): Boolean;`

The **JadeWebServiceProvider** class **isVDFilePresent** method determines whether the file specified in the **fileName** parameter is present in the virtual directory (that is, the directory specified by the value of the **VirtualDirectory** parameter in the **jadehttp.ini** file). This method returns **true** if the specified file exists or **false** if it does not exist.

Note You must call this method during the processing cycle of the message.

***deleteVirtualDirectoryFile* Method**

Signature `deleteVirtualDirectory(fileName: String;
deleteIfReadOnly: Boolean): Integer;`

The **JadeWebServiceProvider** class and **WebSession** class **deleteVirtualDirectoryFile** method deletes files that are in the virtual directory (that is, the directory specified by the value of the **VirtualDirectory** parameter in the **jadehttp.ini** file).

The method parameters are listed in the following table.

Parameter	Description
fileName	Name of the file to be deleted from the virtual directory
deleteIfReadOnly	Deletes files marked as read-only when set to true

This method returns zero (**0**) if the file deletion is successful or it returns a non-zero error code if the deletion fails.

Note You must call this method during the processing cycle of the message.

Handling Web Service Consumer UTF-8 Conversion Errors

In earlier releases, exception 1417 (*Ansi to Unicode conversion failed*) was raised when the Web service client received non-ANSI characters in an ANSI system when calling a Web service.

The **JadeWebServiceConsumer** class now provides the **handleCharConversionException** and **characterConversionException** properties, that enable you to specify not to decode the string when this exception is raised but to process the results using the unconverted string. (Note, however, that the results returned by the framework will contain non-ANSI data.)

Set the **handleCharConversionException** property to **true** if you do not want decoding performed when an exception occurs when decoding a string. When this occurs, the **JadeWebServiceConsumer** class **characterConversionException** property is set to **true** if the string contains non-ANSI characters.

If the **characterConversionException** property is set to **true**, the results will contain non-ANSI characters. The **JadeWebServiceConsumer** class existing **soapResponse** property is set to the returned value so that you can determine the non-ANSI characters (that is, where the character is above 127).

If the **handleCharConversionException** property is set to **false** (the default value), exception 1417 is raised and the **extendedErrorText** property of the exception object is set to the value of the returned result. You can inspect the value of the **extendedErrorText** property, to determine the non-ANSI characters.

Note Exception 1417 occurs in a Unicode system only if the returned result from the Web service contained invalid UTF-8 characters.

Maximizing the Web Service Exposure Wizard

The upper right of the Web Service Exposure Wizard dialog now provides a Maximize button, which enables you to quickly expand the dialog; for example, if the default size is too small to display the full name of the schema entities.

Removing Web Service Consumers

The optional **commandFile** parameter in the batch Schema Load utility (**jadloadb**) now enables you specify a JADE Command File (JCF) that contains the name of a Web service consumer that you want to delete.

The syntax of the command in the JCF is as follows.

```
Delete WebServiceConsumer schema-name::consumer-name
```

The *schema-name* value is the name of the schema in which the Web server consumer you want to delete is defined and the *consumer-name* value is the name of the consumer (that is, the entity displayed in the Web Service Consumer Browser). For example, if the schema is **S1** and the Web service consumer is **WebServiceOverHttpApp**, the command file would look similar to the following.

```
JadeCommandFile
JadeVersionNumber 6.3.05
Commands
AbortOnError False
Delete WebServiceConsumer S1::WebServiceOverHttpApp
```

The success or failure of this command is output to the message log.

Changes in JADE Release 6.3.04

This section contains changes in JADE release 6.3.04.

For details about changes in JADE release 6.3.03 (the first general release of JADE 6.3), see [RelInfo6303.pdf](#), in your JADE **documentation** directory.

Highlights in Release 6.3.04

The following table summarizes the highlights in this release.

Feature	Description
Faster Classes In Use Browser	Faster display of in-use classes for large development systems
Excluding offline objects when iterating a collection	Enhance performance by easily excluding offline objects from collection iterations
Profiler cache statistics	See the size of discarded methods in a profile period
Thin client download process	Avoids unnecessary downloads via stricter file checks

Application Class *doWindowEvents* Method

In earlier releases, the value of the **waitTime** parameter of the **Application** class **doWindowEvents** method was not honored if the message queue contained many callback-type requests (for example, notifications, timers, TCP/IP completion routines, and so on). When the **app.doWindowEvents** method was called, JADE attempted to empty the Windows message queue, regardless of the wait time that was specified. This ensured that all outstanding paint requests were always processed on exit. (Paint requests are generated by Windows only when no other messages are queued.) As a result, a call to **app.doWindowEvents** may have taken considerably longer than the requested time when the queue of callback function requests was large or it was never emptied.

From this release, calling the **Application** class **doWindowEvents** method now results in the processing of queued messages associated with the Graphical User Interface (GUI) and the user interface until:

- The queue is empty *and* the specified wait time has expired (which was the case in earlier releases).
- The specified wait time has expired and another callback-type messages is encountered (the first is always processed).
 - In JADE thin client mode, all other message queue entries are processed and then the callback request is re-posted on exit from the method.
 - In standard client mode, the callback request is processed and the method is exited. In this situation, paint requests and possibly user requests remain unprocessed.

Note The reason for this difference in handling is that in JADE thin client mode, there is only ever one message posted on the presentation client for all outstanding callback-type requests. Conversely, a standard client can have many (that is, hundreds or even thousands).

Classes In Use Browser Performance

The performance of the Classes In Use Browser has been improved by making the load time of classes constant, regardless of the number of processes using the class or classes.

In addition, you can now use the **Show Process Usages** command from the Classes menu when a class is selected in the Class Browser, to view details of the processes using the selected class. For details, see “Browsing Classes that Are in Use”, in Chapter 3 of the *JADE Development Environment User’s Guide*.

Defining a Web Service Exposure

In JADE 6.2, Web services exposures were defined in the Web Services Description Language (WSDL) generation.

In JADE 6.3, Web service exposures must be defined by using the Web Service Exposure Wizard. (For details, see “Using the Web Service Exposure Wizard”, in Chapter 16 of the *JADE Developer’s Reference*.)

Disabled Text Color of a Text Box under Portable GUI

The [Jade] section of the JADE initialization file can now contain the **DisabledTextColor** parameter, which sets the disabled text color of a text box under Portable GUI.

If this parameter is present and the value is not zero (**black**), all disabled text boxes use the specified Integer value to set the color of disabled text in a text box. The default value is zero (**0**); that is, black.

This parameter is read the first time a disabled text box is encountered.

Error Messages

This section describes new error messages in this release.

3115 - GetObject request buffer too small

Exception 3115 is raised when there is a mismatch between the class definitions loaded into memory and the actual definitions in the database. It can also indicate an internal error due to character conversion issues between nodes has occurred.

If a schema has been changed at that time of the exception or shortly beforehand, the problem can be rectified by stopping and restarting the node on which it occurred. This will refresh the class definitions loaded into memory. In these circumstances, this exception is similar to the 3039 exception (*Schema to Database object size mismatch*).

If there were no schema changes around that time, or stopping and restarting does not resolve the problem, please report this to JADE Support.

6428 - Cannot access versioned feature in current version of method

Exception 6428 is raised and a method is marked as being in error if an unversioned method is compiled in the context of the current schema and it references a versioned feature that would result in different compile code.

Excluding Offline Objects when Iterating a Collection

To specify that iterators exclude offline objects, call one of the following methods.

- Call the **Process** class **iteratorsExcludeOfflineObjects** method with the **enable** parameter set to **true**, to cause objects stored in offline partitions to be excluded from the iteration.

This affects explicit collection iterators and **foreach** iterations over object collections executed by the process.

- Call the **Iterator** class **excludeOfflineObjects** method with the **enable** parameter set to **true**, to cause the receiver to exclude objects stored in offline partitions from the iteration.

The exclusion state takes effect on the iteration (that is, the call to the **next** or **back** method) that follows.

JADE Initialization File

For historical reasons, when reading JADE initialization file parameter values, the code in previous releases checked the section name for a **Jade** or **Jom** prefix and ignored the prefix if the section without the prefix was present. For example, for initialization file parameters in the [JadeLog] section, if a [Log] section were present in the JADE initialization file, the [Log] section would have been used instead of the [JadeLog] section.

This check is no longer done.

Object::hasMembers Method

The **Object** class **hasMembers** method, which formerly could be used only with exclusive collections signified to be *delete when emptied*, is no longer condition-safe and cannot be used in constraints. However, you can now use it with any collection.

For exclusive collections that have not been populated or instantiated, this method enables you to determine if the collection is empty without having to access or lock the collection.

Partition Awareness

The Logical Certifier now checks online instances of a class only. However, it can still encounter offline instances when checking inverses.

Patch Number Batch Extract

When performing a batch extract by patch number (for single or multiple schemas), the command file is always created.

For all types of batch extract *other* than an extract by patch number, the command file is not created by default. If you want the command file to be created, you must specify this on the command line; for example:

```
jadclient path=c:\jade\system ini=c:\jade\system\jade.ini
server=singleUser schema=JadeSchema app=JadeBatchExtract endJade Single
c:\temp\myTest.scm c:\temp\myTest.ddb Test "<jcf>"
```

The `<jcf>` value indicates that you want the command file created. As you can specify more than one option between the `<` and `>` characters, you must separate each one with a comma; for example, if you want to encrypt the source and create the command file, your command would look similar to the following.

```
jadclient path=c:\jade\system ini=c:\jade\system\jade.ini
server=singleUser schema=JadeSchema app=JadeBatchExtract endJade Single
c:\temp\myTest.scm c:\temp\myTest.ddb Test "<encrypt,jcf>"
```

Profiler Cache Statistics

Profile cache statistics now include the number of methods that were discarded from the cache to make room for new methods during the profiler run. If there were discarded methods, the total size of the methods discarded is also listed.

Method cache flow statistics are written to the table in the “cache statistics” section of the profiler report when the cache size exceeds the maximum size specified and all methods in the cache were in use and could be discarded.

Relational Population Service (RPS)

The following change has been made to RPS in this release.

Process::*isUserDataPump* Method

The **Process** class now provides the **isUserDataPump** method, which returns **true** if the process is executing as a **Datapump** application on an RPS node; otherwise **false**.

Relational View

The following changes have been made to Relational Views in this release.

Controlling the Size of a Result Set Returned by the ODBC Driver

You can now control the maximum size of a result set to be returned for a call to the **SELECT** statement using the JADE ODBC driver, by specifying the required value in the new **MaximumResultSetSize** parameter in the [JadeOdbc] section of the JADE initialization file.

The default value of zero (**0**) specifies that there is no maximum limit. The value type of this parameter is **Integer64 prefix multiplier** (for example, 16K).

A message is output to the **jommsg.log** if the result set output is truncated because of the value of this parameter.

Property Details Access

The **RelationalView** class now provides the **getColumnFeature** method, which has the following signature.

```
getColumnFeature (tableName: String;
                 columnName: String): Feature;
```

This method returns the feature (method or property) associated with the table column specified by the **tableName** and **columnName** parameters.

Calls to this method can raise the following exceptions.

- `JErr_Table_Not_Found` : Relational Table not found.
- `JErr_Column_Not_Found` : Column not found in Relational Table.

Renaming or Editing User-Defined Table Names

The **RelationalView** class now provides the **change ColumnName** method, which has the following signature.

```
changeColumnName (tableName:      String;
                 oldColumnName: String;
                 newColumnName: String);
```

This method changes the name of a column in the table specified in the **tableName** parameter from the value specified in the **oldColumnName** parameter to the value specified in the **newColumnName** parameter.

Note This method is valid for ODBC relational views only (that is, it is not valid for RPS mappings).

Calls to this method can raise the following exceptions.

- `JErr_Attribute_Name_Conflict` : Attribute name null or already used as column in the selected table.
- `JErr_Invalid_For_RpsMapping` : May only be called for ODBC Relational Views.
- `JErr_Table_Not_Found` : Relational Table not found.
- `JErr_Column_Not_Found` : Column not found in Relational Table.
- `JErr_ColumnName_Cannot_Change` : Column name cannot be changed (for example, oid or index).

The `Jerr_ColumnName_Cannot_Change` global constant is a new global constant in the **JadeOdbc** category.

Release Notes Splash Screen Display

In JADE 6.3.03, the release note splash screen, which displays the major features of the current release and hyperlinks to further information, was displayed every time a system with no user-defined schemas was started, regardless of the value of the **ShowSplashScreen** parameter in the [Jade] section of the JADE initialization file.

From this release, the release note splash screen is not displayed when a system with no user-defined schemas is started if the value of the **ShowSplashScreen** parameter in the [Jade] section of the JADE initialization file is set to **false**.

Reorganization Updates

The **Allow updates** check box on the Classes Needing Reorg dialog is disabled and unselected when the **FastBuildBTreeCollections** parameter in the [JadeReorg] section of the JADE initialization file is set to **true**.

Thin Client Download Process

The automatic JADE thin client download process has been changed so that it now determines if files are identical without relying on their modified timestamps being different.

The automatic download rules are now as follows.

- If the file is not present, the file is downloaded.
- If the file length is different, the file is downloaded.
- If the file is an executable (that is, a file of type **.dll** or **.exe**), the timestamps of when the two files were linked are compared, rather than the timestamps of when the files were modified. (This timestamp is held in the file contents.)
- If the file is not an executable and the modified timestamps of the two files do not match, an MD5 comparison of the file contents is used to determine if the two files are different.

A file download therefore occurs only when the two files are actually different, instead of the timestamps not matching.

In addition, the length of the string value of the **DownloadVersion** and **PreDownloadVersion** parameters in the [JadeThinClient] and [JadeAppServer] sections of the JADE initialization file has been increased from 30 characters to 60 characters.

Web Services

The Web service consumer import routine now loads and saves the **nillable= "true"** attribute.

This change may require you to reload a Web Service Description Language (WSDL) file, as the WSDL import was not previously setting this attribute.

Word Wrapping in Tables

When the values of the **autoSize** and **wordWrap** properties are set to **true**, if the column width has not been set by logic or by the user, the height of the cell is affected only if the value of:

- The **widthPercent** property for the column is greater than zero (0)
- **Table::autoSize** is **AutoSize_Row(1)** and the row height has not been specifically set

If neither of these applies, non-word wrap display is assumed.

If the column width has been set by logic or by the user, the height is automatically sized if the row height has not been specifically set and **Table::autoSize** is one of **AutoSize_Both**, **AutoSize_BothColumnMinimum**, or **AutoSize_Row**.

WSDL Nested complexType Definition Load

Classes built from nested **complexType** definitions in the WSDL schema that do not have an actual name to match on when reloading the WSDL are now saved during the import process with a name derived from the item name and parent name on which the JADE class name is based.

Note If your WSDL schema has nested **complexType** definitions, to enable JADE to save your definitions with derived names and retain your JADE mapping class names, you must import the WSDL into JADE. Subsequent reloading of the WSDL then retains the derived class names.

ZLIB Compression on Compact JADE

For ZLIB compression in single user mode for a device hosting Compact JADE, depending on the model of your Windows Mobile device, the **zlibce.dll** version 1.2.3 or higher may need to be preloaded.

For details, see the **PreLoadLibraryList** parameter in the [JadeEnvironment] section of the *JADE Initialization File Reference*.