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 © 2009 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 8
- Deimplementations and Deprecations ................................................................. 8
- Advanced Notice of ActiveX Exposure Deimplementation ............................. 8
- JADE Portable Graphical User Interface (GUI) Client ....................................... 8
- PKWare Compression ......................................................................................... 9
- Real[10] Parameters in External Function Calls .............................................. 9
- RPS SQL Script Execution .............................................................................. 9

Highlights in this Release 10

Accessing Details about Faults Fixed in Releases 11
- How to Locate PARs Fixed in a Specific Release .............................................. 11

Upgrading to JADE 6.3 from a JADE 6.2.12 Release or Higher 12
- Upgrading to JADE 6.3 from a Windows JADE 6.2.12 Release or Higher .......... 12
- Running Two Windows Releases of JADE on the Same Workstation ............. 14
- Upgrading to JADE 6.3 from a Linux JADE 6.2.12 Release or Higher .............. 15
- JADE Thin Client Upgrade ............................................................................... 15
  - Upgrading a 32-Bit Presentation Client Connecting to a 64-Bit Application Server 16
  - Upgrading a Synchronized Database Environment (SDE) from JADE 6.2 to 6.3 .... 16
  - Upgrade Validation ....................................................................................... 17
  - Hot Fix Releases .......................................................................................... 17

JADE 6.3 Changes that May Affect Your Existing Systems 18
- Compact JADE Support .................................................................................... 18
- Compiler Version ............................................................................................ 18
- Compression and Decompression Methods Deimplemented ......................... 18
- Exception Handling ....................................................................................... 19
  - Cutting Back the Exception Stack ............................................................... 19
  - Exception Dialogs ....................................................................................... 20
  - Initiating an Application ............................................................................. 21
- External Function Calls ................................................................................. 21
- Form::resize Event Method Handling ............................................................ 21
- Handling Delete when Emptied Collections ................................................. 21
- JADE Initialization File Handling .................................................................. 22
- Printing Enhanced Meta Files (EMF) under Windows Vista ........................... 22
- Server Locking when Calling Client Methods ............................................. 22

Changes and New Features in JADE Release 6.3.03 23
- .NET Class Library ........................................................................................ 23
- _userdef.dat File Renamed _rootdef.dat File ................................................ 23
Changes and New Features in JADE Release 6.3.03, continued

64-Bit Edition of JADE .......................................................... 23
Cache Sizes ........................................................................... 24
Identifying Use of 32-Bit or 64-Bit Binaries ......................... 24
Integer64 Versions of Integer Methods ................................ 24
Application Changes ......................................................... 25
Application::getMessageText Method ................................ 25
Running a Superschema Application from a Subschema .... 25
Starting Up Applications .................................................... 25
Application::startApplicationWithString Method .......... 25
Define Application Dialog .................................................. 26
ServerApplication Parameter in the JADE Initialization File .. 26
Class Numbers .................................................................... 26
Process::getTransientOnlyClassesUsed Method .................. 26
Code Maintenance ............................................................. 26
Concurrency Control .......................................................... 27
Update Locks ........................................................................ 27
Exception Handling ......................................................... 28
Lock-Related Constants .................................................... 29
Control and Form Enhancements ....................................... 29
Drawing Optimization ....................................................... 29
Window::beginBatchDrawing Method ............................... 29
Window::endBatchDrawing Method .................................... 30
Folder::topSheet Property .................................................. 30
JadeRichText Control Protected Text .................................. 30
setTextProtection Method ................................................ 30
getTextProtection Method .............................................. 30
protected Event ............................................................. 31
Painting Transparent Controls .......................................... 31
Paging Up and Down a Table Control ................................ 31
Window::getHwnd Method ................................................ 31
Database Backup Process Type .......................................... 31
Database File Partitioning .................................................. 32
JADE Database Administration Utility ............................... 32
JADE Database Utility Changes ......................................... 32
Database Progress Events ................................................ 32
Development and Testing Enhancements ............................ 33
Code Coverage ............................................................... 33
Method Tracking ............................................................ 33
Enabling or Disabling Method Tracking ............................ 34
Transaction Tracing ........................................................ 34
Unit Testing ................................................................. 34
Development Environment Changes ................................. 35
Adding a File that can be Partitioned ................................. 35
Displaying the Release Notes Splash Screen .................... 35
Opening a Class Browser or Wizard for Incomplete Schemas Disallowed .............................................. 35
Opening a New Browser Window ...................................... 36
Executing Conditional Code .............................................. 36
executeWhen Instruction .................................................. 36
JadeExecuteFlagCategory Global Constants Category .......... 36
Changes and New Features in JADE Release 6.3.03, continued

- [JadeExecuteFlags] Section of the JADE Initialization File
- Node Methods
- Identifying Node and Process JADE Distribution
- JADE Database Utility
- Real Primitive Type
- Prohibiting Persistent Transactions and Updates in the Current Process
- Process Class
- Printing
- Object Cache Priority
- Monitoring Enhancements
- MergeIterator
- Loading Schemas
- Localization
- Limit to the Maximum Length of String, StringUtf8, and Binary Primitive Types
- JADE User Interrupt
- JADE Logical Certifier Application
- JADE Database Utility
- Identifying Node and Process JADE Distribution
- SQL Server 2005 and 2008 Default Mappings
- Requirements for RPS Nodes
- Configuring the RPS Node on a Secondary RPS System
- Determining or Assigning a Floating Point
- countQueuedNotifications Method
- number Property
- Prohibiting Persistent Transactions and Updates in the Current Process
- Real Primitive Type
- arcTan2 Method
- Real Primitive Type Constants
- Relational Population Service (RPS)
- Configuring the RPS Node on a Secondary RPS System
- Requirements for RPS Nodes
- SQL Server 2005 and 2008 Default Mappings
- RelationalView Class
## Changes and New Features in JADE Release 6.3.03, continued

<table>
<thead>
<tr>
<th>Feature</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reorganization</td>
<td>46</td>
</tr>
<tr>
<td>Replication Framework</td>
<td>47</td>
</tr>
<tr>
<td>Security</td>
<td>48</td>
</tr>
<tr>
<td>TimeStampOffset Primitive Type</td>
<td>48</td>
</tr>
<tr>
<td>Transient Database File Naming</td>
<td>49</td>
</tr>
<tr>
<td>Translating JadeRichText Context Menus</td>
<td>49</td>
</tr>
<tr>
<td>Web-Enabled Forms</td>
<td>49</td>
</tr>
<tr>
<td>Web Services Extensions</td>
<td>49</td>
</tr>
<tr>
<td>Rich Internet Applications (RIA)</td>
<td>50</td>
</tr>
<tr>
<td>XMI Support</td>
<td>50</td>
</tr>
</tbody>
</table>
Migrating to JADE Release 6.3

This document covers the following topics.

- JADE Release Support
- Highlights in this Release
- Accessing Details about Faults Fixed in Releases
- Upgrading to JADE 6.3 from a JADE 6.2.12 Release or Higher
  - Upgrading to JADE 6.3 from a Windows JADE 6.2.12 Release or Higher
  - Upgrading to JADE 6.3 from a Linux JADE 6.2.12 Release or Higher
  - JADE Thin Client Upgrade
  - Upgrading a Synchronized Database Environment (SDE) from JADE 6.2 to 6.3
  - Upgrade Validation
  - Hot Fix Releases
- JADE 6.3 Changes that May Affect Your Existing Systems
- Changes and New Features in JADE Release 6.3.03

**Tip** For details about using Acrobat Reader to view JADE documents, see “JADE Product Information Library in Portable Document Format”, in Chapter 2 of the JADE Development Environment User’s Guide.

The JADE Product Information Library document (JADE.pdf) provides a summary of contents of documents in the JADE product information library and navigation to the documents.

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.

**Note** To customize the deployment upgrade on Windows, see “Customizing the Deployment Upgrade Process”, in Appendix A of the 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:


For details about the JADE release schedule, go to:

http://www.jadeworld.com/jade/updates.htm

Deimplementations and Deprecations

This section contains the deimplementations and deprecations in this release.

Advanced Notice of ActiveX 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”, later in this document.

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.

As the upgrade to JADE 6.3 will fail if any `Real[10]` parameters are specified, you must change these to `Real[8]` parameters.

The upgrade validation process checks external functions for `Real[10]` parameters and you are alerted of any detected usages are logged, so that you can change them.

RPS SQL Script Execution

In this release, `sqlcmd` has replaced the ODBC interface as the default mechanism for SQL script execution.

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. Use of this utility by RPS requires configuration of the SQL Server instance name 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 the next major release.
# Highlights in this Release

The following table summarizes the highlights in this release.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Benefit</th>
</tr>
</thead>
<tbody>
<tr>
<td>64-Bit Edition of JADE</td>
<td>Takes your JADE systems to new levels of scalability</td>
</tr>
<tr>
<td>.NET Class Library</td>
<td>Extends the reach of your JADE applications</td>
</tr>
<tr>
<td>Database File Partitioning</td>
<td>Improves management of large amounts of historical data</td>
</tr>
<tr>
<td>Update Locks and Mergeliterator</td>
<td>Improves concurrency and performance</td>
</tr>
<tr>
<td>Unit Testing and Code Coverage</td>
<td>Improves the quality of your applications</td>
</tr>
<tr>
<td>Replication Framework</td>
<td>Facilitates the building of disconnected applications</td>
</tr>
<tr>
<td>Multiple Deltas and Patch Control</td>
<td>Improves source control</td>
</tr>
<tr>
<td>XMI Support</td>
<td>Generates schemas from Enterprise Architect (EA) models</td>
</tr>
</tbody>
</table>
Accessing Details about Faults Fixed in Releases

To access the complete documentation about the Product Anomaly Reports (PARs) fixed in this release, run Parsys, our Fault Managements and Customer Contact system. This system also enables you to view the progress of your own contacts.

If you have any queries about Parsys, please direct them to JADE Parsys Support in the first instance, at parsyssupport@jadeworld.com. You can download the install shield for Parsys from the following URL.


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 Parsys 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 locate the PARs fixed in a specific release

1. Select the Advanced Search command from the Search menu with the following settings.
   a. On the Basic Search Criteria sheet, 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 and NFS types are selected.
   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 from a JADE 6.2.12 Release or Higher

This section covers the following topics.

- Upgrading to JADE 6.3 from a Windows JADE 6.2.12 Release or Higher
  - Running Two Windows Releases of JADE on the Same Workstation
- Upgrading to JADE 6.3 from a Linux JADE 6.2.12 Release or Higher
- JADE Thin Client Upgrade
- Upgrading a Synchronized Database Environment (SDE) from JADE 6.2 to 6.3
- Upgrade Validation
- Schema Hot Fix Details

Caution Before you upgrade to JADE 6.3, refer to the “JADE 6.3 Changes that May Affect Your Existing Systems” section, later in this document.

Upgrading to JADE 6.3 from a Windows JADE 6.2.12 Release or Higher

You can upgrade 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 verdist_x86.exe is already installed. If you are upgrading to the 64-bit edition, the Microsoft C++ 2008 SP1 Redistributable Package (x64) called vc_x64Runtime.exe is required. (This executable is supplied on the JADE distribution media.)

Note Installing any Microsoft redistributable package requires administration privileges. If this package needs to be installed, if possible, deploy this package before upgrading to JADE 6.3, using the appropriate techniques that allow for privileged installations.

The JADE Setup program enables you to upgrade your binary and database files to JADE 6.3 from a JADE 6.2.12 release or higher on Windows, by performing the following actions.

1. Ensure that your JADE environment is JADE release 6.2.12 or higher.
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 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 Windows Vista UACs, standard user accounts, and administrator accounts, see:
   
   http://technet2.microsoft.com/WindowsVista/en/library/00d04415-2b2f-422c-b70e-b18ff918c2811033.mspx?mfr=true

4. Take a full backup copy of your existing JADE directories, first ensuring that your database is not in recovery mode.
Upgrading to JADE 6.3 from a JADE 6.2.12 Release or Higher

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. If it is not already installed, the Microsoft Windows C++ Redistributable Package is 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.

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:\my\sys\jade\system\jade.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-Vista operating system or on 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 Vista, the \Program Files portion of the install directory is replaced with 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 Windows 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 recompile any required DLLs.

16. A warning message may be displayed if the upgrade validation process has not completed. If so, check the **jadeupgrade.log** file for information on 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 **jadrap** 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 6.2.12 Release or Higher

You can upgrade 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 6.2.12 or higher.
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.

   ```bash
   /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 Windows Vista UACs, standard user accounts, and administrator accounts, see:


If JADE is installed in the \Program Files directory (or \Program Files (86) directory on a 64-bit machine with 32-bit JADE binaries) when running under Microsoft Windows Vista:

- 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 user name and password of a user with administrative privileges to enable the upgrade to succeed.

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) from JADE 6.2 to 6.3

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.

   **Note** 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.
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:
  1. Binary files (including _sys*.bin and _jad*.bin files)
  2. Monitor and dmpload map files
  3. Reset the timestamps

**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.

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.
JADE 6.3 Changes that May Affect Your Existing Systems

This section describes only the changes in the JADE 6.3 release that may affect your existing systems. Some changes may result in compile errors during the load process, or cause your JADE release 6.3 systems to behave differently.

For details about the changes and new features in JADE release 6.3, see “Changes and New Features in JADE Release 6.3.0.3”, later in this document.

Compact JADE Support

As announced in JADE 6.2.16, Compact JADE Pocket PC 2003 is no longer supported in JADE release 6.3.

JADE release 6.3 does not support Windows Mobile 5.0 for Smartphone, which had been recommended for presentation clients only in earlier releases. The generic category of devices referred to as smart phone must run Windows Mobile 5.0 for Pocket PC Phone Edition.

Compiler Version

JADE 6.3 has been compiled on:

- Windows environments using the Microsoft Visual C++ 2005 compiler for the 32-bit binaries
- Windows environments using the Microsoft Visual C++ 2008 compiler for the 64-bit binaries
- Red Hat Linux environments using the GNU Compiler Collection 4.1.1 (shipped with Red Hat Enterprise Linux 5 Desktop)
- SUSE Linux environments using the GNU Compiler Collection 4.1.0 (shipped with SUSE Linux Enterprise Desktop 10)

If you have any JADE external methods that use the JADE Application Programming Interface (API) calls, you should recompile these methods using the appropriate compiler.

On Windows for the 32-bit binaries, because JADE 6.3 is compiled against Visual Studio 2005, it requires the Microsoft C++ 2005 SP1 Redistributable Package (x86) to be installed. This will be done as part of the normal JADE installation or upgrade. This must also be installed on your IIS server if jadehttp.dll is used.

On Windows for the 64-bit binaries, because JADE 6.3 is compiled against Visual Studio 2008, it requires the Microsoft C++ 2008 SP1 Redistributable Package (x64) to be installed. This will be done as part of the normal JADE installation of the 64-bit binaries. This must also be installed on your IIS server if jadehttp.dll is used.

Compression and Decompression Methods Deimplemented

As announced in the JADE 6.2 release information, JADE 6.3 no longer supports the use of the PKWare compression libraries. The compress methods of types Binary, String, and StringUtf8 and the uncompress, uncompressString, and uncompressStringUtf8 methods of type Binary have been de-implemented in this release.
Before upgrading to JADE 6.3, if you have persistent data that has been compressed using the `compress` method, update that data by uncompressing it by 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.

The use of the compression option represented by the `Binary` primitive type `Compression_PKWare` constant is allowed only for the `compressToBinary` methods of `String`, `StringUtf8`, and `Binary` primitive types if the `UseDeprecatedFeaturePkWare` parameter in the `[JadeEnvironment]` section of the JADE initialization file is set to `true`. Otherwise, an attempt to use this constant will result in a runtime exception.

These changes were made because the PKWare compression libraries are not available for 64-bit platforms. When running JADE 6.3 in 64-bit mode, zlib compression is mandatory. In 32-bit mode, zlib compression will be the default; however PKWare compression will still be available (unsupported and undocumented) for systems that have the `UseDeprecatedFeaturePkWare` parameter in the `[JadeEnvironment]` section of the JADE initialization file set to `true`.

**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 an upgrade to a 64-bit version of JADE can be performed, coding changes to use the new methods specified earlier in this section and data recompression will be necessary.

---

### Exception Handling

The following subsections contain exception handling changes in this release.

#### Cutting Back the Exception Stack

If an exception occurs and an exception handler method catches the exception and does an `Ex_Resume_Next`, the stack is cut back to the method that armed the exception handler and execution continues normally from this point.

However, if a second exception that is also handled by an exception handler that does an `Ex_Resume_Next` occurs while the stack is being cut back, after the stack has been cut back to the method that armed this (second) exception handler, the cutting back of the stack continues to the method that armed the first exception handler.

Consider the following method examples.

```pascal
main() begin
  on Exception1 do handler(exception);  // the handler just does an
    // Ex_Resume_Next
  m1();
  write 'main: returned from m1';
end;
m1();
begin
  m2();
  write 'm1: returned from m2';
end;
```

When Exception1 is raised, the main, m1, m2, m3, and m4 methods are on the call stack. The exception is caught by the exception handler armed in the main method, which does an Ex.Resume_Next. This begins to cut the stack back to the write instruction in the main method.

However when the stack is cut back for the m3 method, a second exception is raised in the epilog code for this method. This exception is also caught by the exception handler, which does another Ex.Resume_Next, which causes the stack to be cut back to the write instruction in the m2 method. When the m2 method has completed, the second exception (Exception2) has now been handled but the first exception (Exception1) is still being handled.

In earlier releases, the cutting back of the exception stack ceased at this point, so the write instruction in the m1 method executed. In JADE 6.3, the stack is cut back over the m1 method so the write instruction in the m1 will not execute, but the write instruction in the main method will execute. Any epilog code in the m1 method will still execute at this point.

**Exception Dialogs**

Under Microsoft Windows, a Window class is created for each kind of window to establish its basic operating style. This class is not visible to you unless you use some type of Windows or user-written utility. For a JADE form, this class has a name of Jade:Form. In earlier releases, a message box and an exception dialog could be created using that JADE form window class, making it difficult for monitoring software to determine if those types of forms are currently displayed.

From this release, exception dialogs are handled as follows.

- When JADE creates a message box as a JADE form (the application is skinned or the application has set the message box caption strings from logic), the class used to create this form will have a Window class name of JadeMsgBox:Form. If a standard Windows message box is displayed, the window class is named #32770 (a standard Windows dialog).

- From JADE 6.2, exception dialogs (that is, standard exception, lock exception, lock retry, and ODBC exception dialogs) are always created as JADE forms and have a Window class name of JadeException:Form.

This change makes it easier for monitoring software to detect if message boxes or exception dialogs are currently displayed.
Initiating an Application

When a JADE application initiates another application (for example, by calling the `startApplication` method) and that application initiation fails, an exception is now always generated by the application requesting the initiation.

In earlier releases, if a non-JADE application called the `startApplication` method and the requested application was not allowed to run on the application server; for example, because of `EnableAppRestrictions` security, no exception was generated. In addition, the process returned by `startApplication` method would have been null, but the reason for the initiation failure would have been lost and no error logged.

External Function Calls

As part of supporting external function calls on 64-bit JADE, the underlying code has been enhanced to allow JADE data types to be more accurately supported. This enhancement may cause an apparent change in behavior. For example, a Windows API that takes a "BOOL" argument should be defined in JADE as an `Integer` primitive type rather than a `Boolean`, because a `Boolean` primitive type is mapped to a C "bool" type and the Windows "BOOL" type is a 32-bit value.

For details about parameter mapping, see “Using External Functions”, in Chapter 5 of your JADE Developer’s Reference.

Form::resize Event Method Handling

The `Form` class `resize` event can result from changes made to a form from your JADE logic; for example, adding or removing a menu item that caused a form `resize` event to be generated.

In earlier releases, this event would be generated during logic execution when running as a standard client or part-way through the update of buffered Graphical User Interface (GUI) changes for a presentation client. This behavior generated issues associated with the JADE Debugger and in thin client mode where the order of GUI updates could result in unexpected results.

From this release, the `Form::resize` event method is now executed only after the current logic processing is complete.

Handling Delete when Emptied Collections

As notified in JADE 6.2 consolidated release information, the Delete When Emptied tuning attribute for collections, set on the `Tuning` sheet of the Define Class dialog, is now deimplemented.

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, it can now be used with any collection. For exclusive collections that have not been populated or instantiated, it provides a means to determine if the collection is empty without having to access or lock the collection.

The upgrade to JADE 6.3 checks for the use of the `hasMembers` method and raises an upgrade error if it is used as a condition.

From this release, you can use the `Collection` class de-instantiation method `deleteIfEmpty` in single user mode only.
JADE Initialization File Handling

As announced in JADE 6.2 consolidated release information, a default JADE initialization file is no longer created if one does not exist.

This change protects against unintentional JADE initialization files being created in the default locations. It also protects against the creation of new files because the name of the initialization file is typed wrongly on the command line. These initialization files can interfere with subsequent JADE thin client automatic upgrades; for example.

If a JADE initialization file does not exist and is required at JADE executable start up, exception 125 is raised (The specified or default Jade INI file does not exist). If a JADE initialization file exists but is empty, JADE writes appropriate default values to it, if required.

**Caution** Ensure that a JADE initialization file exists if you manually install JADE 6.3. (The file can be empty, however.)

A JADE executable locates the JADE initialization file, as follows.

1. A JADE initialization file specified in the ini parameter on the command line, that file is used.
2. No JADE initialization file specified in the ini parameter on the command line but the path parameter is specified, the jade.ini file in the directory specified in the path parameter is used.
3. No JADE initialization file specified in the ini parameter and no path parameter value specified, the jade.ini file in the directory of the executable (for example, bin) is used.

**Recommendation** For JADE executables, specify the fully qualified name of an existing JADE initialization file in the command line.

Printing Enhanced Meta Files (EMF) under Windows Vista

If you use EMF format reports (specified by setting the PrintDataFormat parameter in the [JadePrinting] section of the JADE initialization file to EMF) and then save the reports in the JADE database for future reference or printing, output may be formatted incorrectly because of an incompatibility with the generated EMF report and Windows Vista.

You should therefore regenerate these reports using JADE 6.1.13 or higher, JADE 6.2.14 or higher, or JADE 6.3 and save them again if you plan to upgrade to Windows Vista.

Server Locking when Calling Client Methods

A 1276 exception is now raised when a method running on a client node attempts to update an object that is currently locked due to a request from a serverExecution method, regardless of the duration of the current lock.

In earlier releases, this exception was raised only if the current lock was of session duration.
Changes and New Features in JADE Release 6.3.03

This section summarizes the changes and new features in JADE release 6.3.03. For details about the changes in release 6.3 that may affect your existing systems, see “JADE 6.3 Changes that May Affect Your Existing Systems”, earlier in this document.

.NET Class Library
The .NET class library enables you to expose JADE objects as C# objects, which makes them available to any language in the .NET platform model.

The .NET framework enables you to:
- Use JADE to store and manipulate complex data
- Take advantage of JADE’s automatic object caching and distributed processing

For details, see the JADE .NET Developer’s Reference and Chapter 19, “Using the C# Exposure Wizard”, in the JADE Development Environment User’s Guide.

_userdef.dat File Renamed _rootdef.dat File

The RootSchema default map file _userdef.dat has been renamed to _rootdef.dat so that it no longer shares the _user name prefix with user schema files. This enables you to use a wildcard _user* argument to database utility functions; for example, when you delete user schema files. In earlier releases, you had to explicitly list all files except for the _userdef.dat file.

In addition, if JADE Support requests a copy of user schema files for analysis, you are less likely to supply the renamed _rootdef.dat file, which is seldom required. As this file has become rather large for some sites (for example, 40G bytes in some cases), removing the unnecessary transfer of this file is advantageous.

The _rootdef.dat file contains user persistent instances of RootSchema classes that are not schema entities; for example, shared instances of primitive arrays and OLE objects.

Note When you upgrade to JADE 6.3 from JADE 6.2, the database control file entry and physical file are renamed when the control file is converted.

64-Bit Edition of JADE
The 64-bit JADE implementation enhances performance and scalability by taking full advantage of 64-bit hardware and operating system platforms, as follows.
- As each node has a greater memory address space, administrators can configure significantly larger JADE caches, if required, to improve system performance.
- Internal JADE data structures can now grow beyond the limits imposed by 32-bit hardware and operating system platforms, which provides greater capacity for large JADE systems.

From this release, both 32-bit and 64-bit editions are available on all supported operating systems.

For details about a upgrading a 32-bit presentation client connecting to a 64-bit application server, see “JADE Thin Client Upgrade”, earlier in this document.
Cache Sizes

When using 64-bit JADE, you can set the values of the `ObjectCacheSizeLimit`, `TransientCacheSizeLimit`, and `RemoteTransientCacheSizeLimit` parameters in the `[JadeClient]` and `[JadeServer]` sections of the JADE initialization file to values larger than 2G bytes.

In addition, you can use `Node` class methods to retrieve and set the cache sizes of the node on which the method is executing.

The maximum size that is used is two-thirds of the physical memory size.

Identifying Use of 32-Bit or 64-Bit Binaries

The `Node` class now provides the constants listed in the following table, which can be returned by the `architecture` parameter of the `getOSPlatform` method in the `Node` and `Process` classes if the JADE binaries are built as 64-bit.

<table>
<thead>
<tr>
<th>Node Class Constant</th>
<th>Integer Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture_64Big_Endian</td>
<td>4</td>
<td>64-bit big-endian internal byte ordering and alignment</td>
</tr>
<tr>
<td>Architecture_64Little_Endian</td>
<td>3</td>
<td>64-bit little-endian internal byte ordering and alignment</td>
</tr>
</tbody>
</table>

If the JADE binaries are built as 64-bit, `(x64)` is now appended to the `Server Platform is` string on the JADE About box.

Integer64 Versions of Integer Methods

JADE now provides `Integer64` versions of `Integer` methods on the `Node` class and the `Collection`, `File`, and `Iterator` classes and subclasses. For details, see Volume 1 and Volume 2 of the JADE *Encyclopaedia of Classes*.

The signatures of a number of methods on the `Array` class have been changed. The `index` parameter has changed from `Integer` to `Integer64`. This change does not affect your existing code, as the underlying methods have been changed to accept either an `Integer` or an `Integer64` parameter value.

Although the parameter is an `Integer64` value, collections cannot contain more than 4,294,967,295 entries.

The following table lists the methods whose `index` parameter is now an `Integer64` value.

<table>
<thead>
<tr>
<th>Class</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>Array</td>
<td>at</td>
</tr>
<tr>
<td>Array</td>
<td>atPut</td>
</tr>
<tr>
<td>Array</td>
<td>initialise</td>
</tr>
<tr>
<td>Array</td>
<td>insert</td>
</tr>
<tr>
<td>Array</td>
<td>removeAt</td>
</tr>
<tr>
<td>Array</td>
<td>replace</td>
</tr>
<tr>
<td>ArrayIterator</td>
<td>startAtIndex</td>
</tr>
<tr>
<td>ArrayIterator</td>
<td>startNearIndex</td>
</tr>
<tr>
<td>DictIterator</td>
<td>startAtIndex</td>
</tr>
<tr>
<td>DictIterator</td>
<td>startNearIndex</td>
</tr>
<tr>
<td>ExternallIterator</td>
<td>startAtIndex</td>
</tr>
</tbody>
</table>
Application Changes

This section contains application-related changes in this release.

**Application::getMessageText Method**

The Application class now provides the getMessageText method, which returns the error text associated with an exception specified in the msgNumber parameter. Use this method when you know the errorCode value but do not have the exception instance.

**Running a Superschema Application from a Subschema**

The Run Application dialog now contains the Show Inherited check box, which is unchecked by default.

When you check the Show Inherited check box, the Application Name combo box is then populated with the applications from the current schema and its superschemas.

**Starting Up Applications**

In earlier releases, the Application class startApplicationWithParameter method enabled you to pass a shared transient or a persistent object reference to the initialize method of an application.

You can now start an application by passing a String parameter value to any method in the application that has a single String parameter.

**Application::startApplicationWithString Method**

The Application class startApplicationWithString method enables your logic to initiate another application on the same node as the initiating application and to pass a single string to the initialize method of the new application. The initialize method must have a signature that contains only the initializeParameter parameter.

This method has the following signature.

\[
\text{startApplicationWithString}(\text{schemaName}: \text{String}; \text{appName}: \text{String}; \text{initializeParameter}: \text{String}) : \text{Process};
\]

This method returns the process of the application that was started. The application that calls the method continues executing after JADE has successfully created a new process.

The following code fragment shows the use of the startApplicationWithString method.

```java
app.startApplicationWithString("BankSystem", "UpdateCustomer", "ZQ32112");
```
Define Application Dialog

The Initialize Method combo box on the Define Application dialog now lists all methods in the application that have a single String type or object reference parameter.

See also “Web-Enabled Forms”, later in this document.

ServerApplication Parameter in the JADE Initialization File

The ServerApplication parameter in the [JadeServer], [JadeAppServer], and [NonGuiClient] sections of the JADE initialization file now allows you to specify the initialize method of the target application.

You can specify the optional initialize method and its String parameter value or its object reference after the optional Integer start time value, as shown in the following examples.

```
[JadeServer]
ServerApplication1 = ExampleSchema,ExampleApp,2359,initialize
ServerApplication2 = NoteSchema,NoteApp,,setApplictionSkin(null)
```

The optional initialize method parameter has a maximum length of 250 characters and cannot contain commas.

Class Numbers

Following the provision of transient-only classes in JADE 6.2, the connection between the class numbers and the bit pattern for transient-only classes is allocated dynamically and can be different for each process. (Transient objects are available only to the process that created them.) For details about transient objects, see “Managing Transient Cache Space” under “JADE Object Handling”, in Chapter 1 of the JADE Object Manager Guide and “Transient-Only Classes”, in Chapter 1 of the JADE Development Environment User’s Guide.

Transient-only classes have class numbers in the range 20,480 through 31,279, providing 10,800 transient-only classes, with each process limited to 4,096 transient-only classes in use at any time.

Process::getTransientOnlyClassesUsed Method

The Process class now provides the getTransientOnlyClassesUsed method, which returns the number of transient-only classes being used by the process.

Note Each process is limited to 4,096 transient-only classes in use at any time.

Code Maintenance

In addition to the read-only system database map files documented under “System Map Files”, in Chapter 3 of the JADE Development Environment User’s Guide, JADE now provides the _systools.bin file, which contains meta data objects defined in the JadeMonitorSchema schema and the JadeDumpLoadSchema schema.

This separation of JadeSchema and system tools data enables the delivery of early features in future releases that you can deploy in isolation because JADE development, run time, and tools are now physically isolated.
The system files are as follows.

- JADE development environment system files are _sysdev, _jadeapp, and _jadedef
- Tools system files are _systools, _dmpload, and _monitor
- Deployment system files are _system, _sysgui, _sysxrf, _sysint, _rootdef, _stats, and _sysdef

When deploying a JADE system, you can optionally:

- Mark development system files offline, if you want to prevent users from using the JADE development environment.
- Mark development system files offline and not deploy them, to prevent users from using the JADE development environment. When the JADE development environment is required, restore these files in the deployed system and mark them as online.

**Note**  The separation of runtime and development files is not intended to stop unlicensed users running the JADE development environment but to enhance development mid-release without affecting run time environments.

However, if a JADE system had a production (non-development) licence applied, JADE will not allow you to apply a free developer licence. If the JADE system already has a development licence, users can apply the additional development system files.

- For Compact JADE on mobile devices, the development and tools system files are not installed.
- When providing a system to customers, mark the development system files offline and do not include them in the installer. This prevents customers from using the JADE development environment and other tools.

**Notes**  Marking _sysdev, _jadeapp, and _jadedef system files offline prevents the JADE development environment from being used.

Marking _systools, _monitor, and _dmpload system files offline prevents the JADE Monitor, JADE Dump and Load utility, code coverage, and XML Metadata Interchange (XMI) from being used. For details, see the jdbutilb batch JADE Database utility markOffline command and markOnline command in Chapter 1 of the JADE Database Administration Guide.

### Concurrency Control

This section summarizes the concurrency control enhancements in this release that can help you to reduce locking contention in your JADE systems; for example, persistent collection bottlenecks in systems with a large number of transactions.

#### Update Locks

JADE now provides an update lock, with strength between a reserve and an exclusive lock. Whereas an exclusive lock blocks shared locks, an update lock allows updates but is compatible with shared locks; that is, an update lock allows a single writer and multiple readers, because of transaction isolation.

The process with an update lock can update the object. Other processes can access the object with shared locks, but they see the most-recent committed edition.
Update locks:
- Apply only to persistent objects
- Can be used only when in persistent transaction state
- Can be used only with collections

Enable update locks by calling the Process class `useUpdateLocks` method, which has the following signature.

```java
useUpdateLocks(b: Boolean);
```

Set the `b` parameter to `true` if you want the automatic lock applied when an object is first updated to be an update lock rather than an exclusive lock. Set the parameter to `false` if you want to disable the use of the update lock.

The following table lists lock compatibility.

<table>
<thead>
<tr>
<th>Lock Type</th>
<th>Shared</th>
<th>Reserve</th>
<th>Update</th>
<th>Exclusive</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shared</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Reserve</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Update</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Exclusive</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

Before the process commits the transaction, update locks are automatically upgraded to exclusive locks, by blocking all subsequent shared locks and then upgrading each update lock. (An update lock cannot be upgraded until all existing shared locks are released.)

When an update lock is being placed and the process already has a shared lock on the object, the shared lock is released before the update lock is requested. This occurs even when the process is in transaction state. In earlier releases, locks were not released when in transaction state, even if requested.

Releasing the shared lock allows the possibility that another process updates the object before the update lock is acquired. A continuable exception is raised if this has happened; that is, the object has been updated before the update lock is acquired.

Shared locks are released to avoid an upgrade deadlock, which can happen when two processes that have shared locks on the same object both attempt to update it.

Releasing the shared lock can be avoided by explicitly acquiring the update lock before using the collection; that is, before the shared lock would otherwise be acquired.

### Exception Handling

Because update locks are upgraded to exclusive locks before committing transactions, object locked and deadlock exceptions can occur. Object locked exceptions are reported as normal lock exceptions (that is, error code 1027), which allows exception handlers to determine the object involved, to retry the lock, and to abort or continue the lock, as required. You should retry an exclusive lock for the object involved.

When a lock exception is continued, the commit carries on, upgrading the remaining update locks to exclusive locks and then committing the updates.

If the lock exception handler attempts to continue without successfully retrying the lock, a 1225 (Lock cannot be continued) exception is raised. If the lock exception is not successfully continued, the transaction is automatically aborted.
Use the Process class isCommitting method to determine if a process is currently committing. This method returns true if the process is currently committing a transaction.

Deadlock exceptions during commit are normal deadlock exceptions (that is, error code 1081), with standard exception information available.

When the lock on an object is upgraded from shared to update (for example, when a collection is iterated then updated while in transaction state), the object is unlocked before the update lock is requested, so that a different process can update the object before the update lock is acquired.

If the object were to be updated, exception 1146 (The object was updated before the lock upgrade completed) is raised. This exception is continuable; that is, the exception handler can return Ex_Continue, allowing the commit to carry on and take place. This can be done if it does not matter if the object was updated before the update lock could be acquired.

**Lock-Related Constants**

The Locks category now provides the global constants listed in the following table.

<table>
<thead>
<tr>
<th>Global Constant</th>
<th>Integer Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Get_Lock</td>
<td>0</td>
<td>Occurs when a process requests an exclusive lock on a stable object, but is queued because the object is currently shared locked. It can also occur on SDS secondaries when a replaying transaction wants an object locked by an application running on the node. A lock entry with type 0 is created, to indicate that the process does not yet have the lock but all requests from other processes should be blocked and queued until the process gets the lock.</td>
</tr>
<tr>
<td>Update_Lock</td>
<td>4</td>
<td>Manually acquires the Update lock before using the collection; that is, before the Share lock would otherwise be acquired.</td>
</tr>
</tbody>
</table>

In addition, the Lock class now provides the Type_Update and Type_Get constants, which have Character values of '00' and '04', respectively.

**Control and Form Enhancements**

This release provides the control and form enhancements summarized in the following subsections. For full details, see Chapter 2 of the JADE Encyclopaedia of Classes.

**Drawing Optimization**

The Window class now provides two new methods, which provide the ability to store the drawing requests called on a Window without performing the actual drawing until after all of the requests have been issued. This enhancement:

- Removes any flicker associated with the drawing
- Provides faster performance, as the actual drawing can re-use the same Windows drawing environment for the entire script

**Window::beginBatchDrawing Method**

The beginBatchDrawing method starts the batch mode operation. All subsequent drawing requests are collected into the drawing script but are not drawn.
Exception 14258 is raised if the value of the autoRedraw property is not true for this window when this method is called.

Window::endBatchDrawing Method

The endBatchDrawing method cancels the batch mode operation and requests that the window redraw itself so that the drawing script will be replayed.

If the window is repainted while still in batch mode, batch mode is cancelled and another paint request is requested in case the whole window is not being redrawn.

Folder::topSheet Property

If the value of the Folder class topSheet property is changed and the control with focus is not a child of the folder, focus is not moved.

JadeRichText Control Protected Text

The entities summarized in the following subsections enable you to mark a selection of content in a JadeRichText control as being protected. Protected text cannot be changed by users.

setTextProtection Method

The setTextProtection method, which has the following signature, enables you to mark content of a JadeRichText control from the position specified in the start parameter through to the length specified in the length parameter as protected or unprotected, by setting the protected parameter to true or false, respectively.

setTextProtection(start:   Integer;  
length:   Integer;  
protected: Boolean);

Exception 14217 (This feature is not available in this environment) is raised when this method is called in a portable GUI environment.

getTextProtection Method

The getTextProtection method, which has the following signature, returns the protection state of text from the position specified in the start parameter through to the length specified in the length parameter.

The return value indicates if the text is protected, unprotected, or contains a mixture of protected and unprotected text.

getTextProtection(start:  Integer;  
length: Integer): Integer;

The return values, represented by new JadeRichText class constants, are listed in the following table.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Integer Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TextProtection_Set</td>
<td>1</td>
<td>All text in the specified range is protected</td>
</tr>
<tr>
<td>TextProtection_NotSet</td>
<td>0</td>
<td>No text in the specified range is protected</td>
</tr>
<tr>
<td>TextProtection_Mixed</td>
<td>#80000000</td>
<td>Mixture of protected and unprotected text</td>
</tr>
</tbody>
</table>

Exception 14217 (This feature is not available in this environment) is raised when this method is called in a portable GUI environment.
**protected Event**

The `protected` event, which has the following signature, is raised whenever a user attempts to alter in any way (for example, to insert, delete, or change the size of the font) text that is marked as protected.

```plaintext
protected(textbox:     JadeRichText;
    start:       Integer;
    length:      Integer;
    allowChange: Boolean io);
```

Your logic can determine if changes the rich text control should be allowed. Set the value of the `allowChange` parameter to `true` to allow the changes or to `false` if changes cannot be made to text in the control. The values of the `start` and `length` parameters indicate the block of text that the user is attempting to change.

If you do not implement this event on a specific control, any attempt to alter protected text is ignored.

In JADE, you can set or clear text protection only via logic (that is, by calling the `setTextProtection` method). Protected text in rich text format created externally to JADE and loaded into the `JadeRichText` control is now honored by JADE.

When you call the `setTextProtection` method with the `protected` parameter set to `true` on protected text, the `protected` event is not raised.

**Painting Transparent Controls**

It not possible to handle the painting of transparent controls in the correct `zOrder` when it involves a mixture of controls that can be directly painted by JADE and those that can only be painted separately by Windows.

As a result, transparent sibling controls are always be painted before any `JadeRichText`, `MultiMedia`, `JadeXamlControl`, `Ocx`, `OleControl`, `JadeDotNetVisualComponent`, or `ActiveXControl` controls, regardless of their `zOrder` settings.

**Paging Up and Down a Table Control**

Pressing the PAGE UP or PAGE DOWN key when the focus is in a text box that is a `cellControl` property on a `Table` control now passes the page up or page down action to the table and no longer to the text box. (This behavior already applies if the cell control is a `JadeEditMask` control.)

This behavior does not apply to any other cell control type. For a `ComboBox` control, for example, the PAGE UP or PAGE DOWN keys affect the combo list that is displayed.

**Window::getHwnd Method**

The `Window` class now provides the `getHwnd` method, which returns the Microsoft Windows handle for a form or control.

**Database Backup Process Type**

The value of the `Process` class `type` property for a database backup is now `6` (a JADE tool, run as an application from the JADE development environment; for example, the JADE Monitor).
Database File Partitioning

Database file partitioning provides:

- Easier management of large amounts of historical data, particularly for bulk data operations such as backups and deletions
- Cost savings through the use of tiered disk storage, which enables you to split database files into partitions, with different partitions then being able to be stored on different media devices, tiered disk storage, or both different media devices and tiered disk storage
- The ability to move objects between partitions programmatically enables you to avoid accessing historical information if it is not required, which improves performance of online and batch operations

A logical database file can be split into several parts, or partitions, where each partition corresponds to a physical file at the file system level. A file can have a maximum of 256 partitions.

For details, see Chapter 25, “Partitioning Database Files”, of the JADE Developer’s Reference.

JADE Database Administration Utility

JADE now provides the batch JADE Database Administration utility (the jdbadmin program), which enables you to execute database administrative operations, including new file partition-related operations, online in single user mode or in multiuser mode.

For details, see Chapter 2 of the

JADE Database Utility Changes

The batch JADE Database utility (the jdbutil program) backup, restore, and restoreFile commands can now have optional excludeFrozenFiles and excludeFrozenPartitions parameters, which default to false.

The JADE Database utility (the jdbutil program) Backup Database, Restore Database, and Restore Database And Roll Forward dialogs provide the Exclude Frozen Files and Exclude Frozen Partitions check boxes, which are unchecked by default.

For details, see Chapter 1 of the JADE Database Administration Guide.

Database Progress Events

The DbFile class BackupOperationEvent event userInfo parameter passed to the notification callback has changed content format. In earlier releases, this string could contain the "Backup", "Backup and Compress", or "Verify Checksum" values.

From this release, this string contains an operation and filename pair that has the following format.

"operation=operation-text;fileName=filename-text"

The DbFile class now provides for three additional events, which have the class constant values of BackupBytesDoneEvent, BackupOutputEvent, and BackupErrorEvent and which can be subscribed to during the backup of a file.

- The BackupErrorEvent event userInfo parameter passed to the notification callback is an integer error value. This provides for inclusion of error information in output streams of backup applications.
The **BackupOutputEvent** event **userInfo** parameter passed to the notification callback is a string containing information pertinent to the backup operation for inclusion in output streams of backup applications. For example, if a partitioned file an offline partition is encountered when backing up, the "**Skipped offline file partition-fileName-text**" message will be received.

The **BackupBytesDoneEvent** event **userInfo** parameter passed to the notification callback is a string containing the operation information and the progress made thus far in terms of bytes.

The string has the following format, in which the *byte-count-value-text* values are formed from 64-bit file offset values.

```
"operation=operation-text;fileName=filename-text;
bytesDone=byte-count-value-text;bytesToDo=byte-count-value-text"
```

As with the existing **BackupProgressEvent** event, the **DbFile** class **BackupBytesDoneEvent** notification is generated only if the backup application has enabled byte progress event notifications. The **JadeDatabaseAdmin** class now provides the **enableByteProgressEvents** and **disableByteProgressEvents** methods for this purpose.

The **enableByteProgressEvents** method takes an integer increment parameter that specifies the byte value increment to be used for progress reporting (a zero value defaults to 128K bytes).

For details, see Volume 1 of the JADE *Encyclopaedia of Classes*.

**Development and Testing Enhancements**

This section summarizes the development and testing enhancements in this release.

**Code Coverage**

*Code coverage* describes the degree to which the methods in a system have been tested. It enables you to assure the quality of a set of tests, as opposed to directly reflecting the quality of the system under test. Code coverage can help testers and developers to:

- Discover methods and blocks of code that are not exercised by a set of tests
- Create tests that increase code coverage
- Quantify the overall code coverage of a system, which is one measure of quality
- Analyze JADE methods, to determine those lines of code that have been executed or have not executed at all

Code coverage shows lines that have been executed or not executed at all.

For details, see “*Code Coverage*”, in Chapter 22 of the JADE *Developer's Reference* and “Determining Code Coverage” under “Using the JADE User Interrupt”, in Chapter 1 of the JADE *Run Time Application Guide*.

**Method Tracking**

JADE now provides method tracking, which enables you to specify methods to be automatically invoked just before a specified method is called and just after the method has returned execution, without requiring any code changes to the method being tracked so that you can intercept calls to specified methods.
From the method that is invoked, you can find out the name of the tracked method, its receiver object, its input and output parameters, and its return value.

For details, see Chapter 23, “Tracking Methods”, in the JADE Developer’s Reference.

**Enabling or Disabling Method Tracking**

The [JadeSecurity] section of the JADE initialization file on the server node now provides the MethodTrackingEnabled parameter, which defaults to false, and which is checked when the Process::startMethodTracking method is first called (but not for any subsequent calls) to that method. By reading the JADE initialization file when checking, the value can be temporarily overwitten without having to restart the server node.

To enable system-wide method tracking, set this parameter to true. You can change the value of this parameter at any time; that is, you do not have to stop and restart the database server node.

When set to false, any attempt to use the Process::startMethodTracking method raises an exception.

**Transaction Tracing**

JADE now provides transaction tracing, which enables you to determine which objects have been updated, created, and deleted within a transaction. For details, see Chapter 24, “Tracing Transactions”, in the JADE Developer’s Reference.

**Notes**

Transaction tracing applies only to persistent transactions.

You can specify multiple call back methods.

The Process class provides methods that enable, start, and stop transaction tracing. For details, see Volume 2 of the JADE Encyclopaedia of Classes.

When transaction tracing is enabled, information on objects updated, created, and deleted by the current transaction is placed in a transient instance of the new JadeTransactionTrace class. You can examine this instance later, to access the list of objects. For details, see Volume 2 of the JADE Encyclopaedia of Classes.

An application can specify one or more methods to be automatically invoked immediately prior to a transaction being committed.

There is one automatic active trace object for each process.

**Unit Testing**

Unit testing enables you to improve the quality and reliability of your applications by writing unit tests as your system is built and by performing extensive unit testing with minimal user intervention.

You can develop tests for a unit of code (which can be a fragment, a method, or a module) as the code is written. Your development teams can employ this approach to ensure that quality is built into a project from the early stages.

Unit tests isolate each part of the program and show that the individual parts are correct. They provide a strict, written contract that the piece of code must satisfy. Use unit testing to:

- Refactor at a later date, to ensure that the module still works correctly (that is, for regression testing)
- Eliminate uncertainty in the units themselves and can be used in a bottom-up testing style approach
Provide living documentation of the system, which can provide a developer with a basic understanding of the functionality and use of a unit.

JADE provides a framework to enable the automatic execution of these tests, as well as providing the ability to browse tests and extract results. You can build your own unit testing driver (progress tool), instead of using the default JADE unit test functionality.

You can execute the unit testing framework by batch execution of the tests and from the JADE development environment.

For details about using the JADE testing framework, see Chapter 22, “Using the JADE Testing Framework”, in the JADE Developer’s Reference, “Running a Unit Test”, in Chapter 4 of the JADE Development Environment User’s Guide, and the JadeTestCase class and JadeTestListenerIF interface, in Volume 2 of the JADE Encyclopaedia of Classes.

Development Environment Changes

The following subsections summarize changes to the JADE development environment in this release. For details, see the JADE Development Environment User’s Guide. See also “Comparing Method Sources” and “Merging Compared Source” under “Patch Control”, later in this document.

Adding a File that can be Partitioned

The File dialog (accessed by clicking the Add Map File or Add Default Map button on the Add Schema dialog) now provides the Partitionable check box, which is unchecked by default. Check this check box if the database map file can be partitioned.

A partitionable database file must have zero or one classes mapped to the file. A collection class cannot be mapped to a partitionable file.

You cannot change the value of the Partitionable check box if the DbFile class isPartitioned method returns true.

Displaying the Release Notes Splash Screen

The release note splash screen, which displays the major features of the current release and hyperlinks to further information, is now displayed:

- The first time an upgraded JADE system is started, regardless of the value of the ShowSplashScreen parameter in the [Jade] section of the JADE initialization file.
- The first time a new installation of JADE is started.
- Every time a system with no user-defined schemas is started, regardless of the value of the ShowSplashScreen parameter in the [Jade] section of the JADE initialization file.

You can then check the Do not show at startup check box on the release note splash screen if you want to suppress the display of the screen the next time the JADE development environment starts up.

Opening a Class Browser or Wizard for Incomplete Schemas Disallowed

The Schema Browser now highlights an incomplete schema (that is, one in which circular package references remain unresolved) with a default background color of red. (You can use the Window sheet of the Preferences dialog to set this background color to a color of your choice.)
If you select an incomplete schema in the Schema Browser, the Schema, Browse, and Jade menus are disabled. If you have selected an incomplete schema and you then attempt to open a Class, Primitive Type, Map, or Application Browser by clicking on the relevant toolbar button, a message is displayed, advising you that the schema is incomplete.

**Opening a New Browser Window**

The SHIFT+F11 function key combination now opens a new browser window in the schema in which the entity under the caret is defined.

**Executing Conditional Code**

You can now conditionally execute blocks of code, by using the functionality summarized in the following subsections.

**executeWhen Instruction**

JADE now provides the `executeWhen` instruction, which enables:

- Blocks of code to be executed only if the value of a condition is `true`
- Blocks of code not to be loaded if the value of a condition is `false`
- The value of the condition to be changed dynamically while the application is running

The syntax of the `executeWhen` instruction is shown in the following example.

```plaintext
executeWhen DebugTestFlag;
write cust.balance;
endExecuteWhen;
```

For more details, see “executeWhen”, in Chapter 1 of the JADE *Developer’s Reference*.

**JadeExecuteFlagCategory Global Constants Category**

JADE now provides the `JadeExecuteFlagCategory` global constants category, in which you add the global constant that you want to conditionally execute.

In the syntax example under “`executeWhen` Instruction” in the previous subsection, `DebugTestFlag` is a `Boolean` constant added to the `JadeExecuteFlagCategory` global constants category. You must first add this global constant category in your user schema or a superschema, as it is not defined in the `RootSchema`.

The `executeWhen` conditional instruction does not depend on the defined `true` or `false` value of the global constant. The defined value is always ignored when an `executeWhen` condition is evaluated.

**[JadeExecuteFlags] Section of the JADE Initialization File**

The effective value of the global constant in the `JadeExecuteFlagCategory` global constants category is read from the `[JadeExecuteFlags]` section of the JADE initialization file when the node is initialized, as shown in the following example.

```plaintext
[JadeExecuteFlags]
DebugTestFlag=true
```
If the [JadeExecuteFlags] section does not contain a parameter corresponding to a global constant, the effective value of `false` is used in an `executeWhen` instruction.

**Note** Although the defined value of a global constant never changes, its effective value used in an `executeWhen` instruction can change.

### Node Methods

The `Node` class now provides the `getExecuteFlagValue`, `setExecuteFlagValue`, and `clearMethodCache` methods.

The `setExecuteFlagValue` method enables you to change the effective value of a global constant used in an `executeWhen` conditional instruction, overriding the value specified in the [JadeExecuteFlags] section of the JADE initialization file.

You should also call the `clearMethodCache` method, as the `setExecuteFlagValue` method could have been loaded previously, as shown in the following code fragment.

```java
node.setExecuteFlagValue("DebugTestFlag", false);
node.clearMethodCache;
```

The `getExecuteFlagValue` method returns the current, effective value of a global constant used in an `executeWhen` conditional instruction.

### Identifying Node and Process JADE Distribution

The `Node` and `Process` classes now provide the `getOSDetails` method, which has the following signature.

```java
getOSDetails(jdo: JadeDynamicObject input);
```

This method enables you to determine the various usages of JADE for a specific environment; for example:

- On a platform that returns operating system of `OSUnixLinux`, architecture of `Architecture_64Big_Endian`, and version of `2.6`, this method can provide a distribution of `x86-64-redhat-rhel5` as distinct from `i686-redhat-rhel5` or `i686-suse-sles10`.
- Type of binaries that are required for thin client downloads (for example, `x64-msoft-win64-ansi`).

For details about the values that populate the JADE dynamic object parameter, see the `Node` or `Process` class, in Volume 2 of the JADE *Encyclopaedia of Classes*.

### JADE Database Utility

The batch JADE Database utility (`jdbutilb`) now provides the `markOffline` and `markOnline` commands, which enable you to change the state of database files or selected files to offline (for example, to mark the development-only `_sysdev`, `_jadapp`, and `_jadedef` system files as offline in a non-development database) and to change the state of offline database files or selected files to online, respectively.

See also “JADE Database Utility Changes” under “Database File Partitioning” and “Code Maintenance”, elsewhere in this document.
**jdbutilb** verifyChecksums Command Multiple Worker Threads

The batch JADE Database utility (**jdbutilb**) now enables you to optionally specify the `workers` parameter if you want to use a single worker thread (1) or more than the default two (2) multiple concurrent worker threads.

The **verifyChecksums** operation can use a maximum of 16 multiple concurrent worker threads.

The following is an example of the checksum verification command.

```
jdbutilb path=d:\jade6309\backup\offline\compressed verifyChecksums workers=3
```

**JADE Initialization File**

This section summarizes the changes to the JADE initialization file in this release. For details, see the JADE Initialization File Reference. See also “Executing Conditional Code” and “JADE Initialization File Handling”, elsewhere in this document.

**[JadeSound] Section Deimplemented**

The [JadeSound] section of the JADE initialization file is no longer used, as the use of sound to highlight client node connections and disconnections has been deimplemented.

**Node Name Parameters**

The **NodeNameDescription** and **NodeNameFriendly** parameters in the [JadeServer] section of the JADE initialization file are now also used by the database server node (that is, **jadrap** or **jadserv** on Windows or **jad** on UNIX).

In earlier releases, these parameters were used only by the **jadapp** and **jadappb** application server executables.

**Reorganization Parameters**

The JADE initialization file can now contain the [JadeReorg] section, which contains the following parameters that store reorganization settings.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>FastBuildBTreeCollections</strong></td>
<td>false</td>
</tr>
<tr>
<td><strong>ReorgWorkDirectory</strong></td>
<td>&lt;default&gt;</td>
</tr>
<tr>
<td><strong>ReorgSortDirectory</strong></td>
<td>&lt;default&gt;</td>
</tr>
<tr>
<td><strong>ReorgBackupDirectory</strong></td>
<td>&lt;default&gt;</td>
</tr>
<tr>
<td><strong>ReorgWorkerThreads</strong></td>
<td>1</td>
</tr>
<tr>
<td><strong>WorkerSortMemory</strong></td>
<td>50M</td>
</tr>
</tbody>
</table>

Although the **ReorgBackupDirectory**, **ReorgWorkDirectory**, and **ReorgWorkerThreads** parameters could be specified in the [PersistentDb] section in earlier releases, the preferred location of these parameters is now the [JadeReorg] section. (If these parameters are specified in both the [PersistentDb] and [JadeReorg] sections, the parameters in the [JadeReorg] section have precedence when a reorganization is initiated.)

See also “Reorganization”, later in this document.
JADE Logical Certifier Application

The Jade Logical Certifier dialog now provides the **Certify Created During Date Range** option button. When you select this option, you can enter the start and end creation dates of the items that you want certified. The **JADE Logical Certifier** application then checks only the items that were *created* within the specified date range (that is, updated instances are not certified).

By default, all items in the selected operation are certified.

JADE User Interrupt

This section summarizes the JADE User Interrupt changes in this release. For details, see “Using the JADE User Interrupt”, in Chapter 1 of the JADE *Runtime Applications Guide*.

Code Coverage

The JADE User Interrupt now provides the Code Coverage menu, which enables you to start and stop code coverage and to report or display the lines of code in your methods that have been executed or not executed at all.

See also “Code Coverage” under “Development and Testing Enhancements”, earlier in this document.

Display of the Instance Identifier

Each application listed in the JADE Interrupt menu now includes the instance identifier (**instId**) of the associated JADE process.

This identifier is the same as that shown in the JADE Monitor Users view, and it enables you to distinguish between multiple copies of the same application running on the same presentation client (for example, if you want to determine the user who acquired a selected lock so that you can force off or interrupt the user).

Limit to the Maximum Length of String, StringUtf8, and Binary Primitive Types

JADE limits the maximum length of a:

- **Binary** primitive type value to 2,147,483,647 bytes
- **String** or **StringUtf8** primitive type value to 2,147,483,647 characters

**Note** This is documented for clarification purposes only; there has been no change to the limits or behavior in this release.

Loading Schemas

This section summarizes the changes to the JADE Schema Load utility (**jadload**) and the batch JADE Schema Load utility (**jadloadb**) in this release.

The Load Schema dialog now provides a **More Options >>** button, which extends the dialog and displays the Advanced Parameters and System Parameters group boxes. The caption on the button in the extended Load Schema dialog is **Hide Options <<**.
In addition to controls provided by the Load Schema dialog in earlier releases, the Advanced Parameters group box contains the following controls.

- **Allow Circular Packages** check box
- **Compile Unchanged Methods** check box
- **Create Delta if Missing** check box
- **Delta Name** text box, which enables you to specify the name of the delta into which all loaded methods will be checked out
- **Ignore Empty Methods** check box
- **User Name** text box, which enables you to specify the user name for the delta

The check boxes are unchecked by default. If you check the **Create Delta if Missing** check box, you must specify a valid user name and a delta name in the respective text boxes.

The extended Load Schema dialog now provides the following combo boxes, which have a default value of **Confirm**.

- **Delete Methods If Absent** combo box
- **Delete Properties If Absent** combo box

The value that you specify or select in the existing **Delete If Absent** combo box populates the **Delete Properties If Absent** and **Delete Methods If Absent** combo boxes. You can then select or specify a different value (that is, **Never** or **Always**) for these elements, if required.

For more details, see “Loading a Schema and Forms into a JADE Database”, in the JADE Schema Load Utility User’s Guide.

The **jadloadb** batch JADE Schema Load utility now provides the following commands.

- `allowCircularPackages`
- `createDeltaIfMissing`
- `delta`
- `userName`

For details, see “Loading a Schema and Forms in Batch Mode”, in the JADE Schema Load Utility User’s Guide.

### Localization

This section summarizes the localization enhancements in this release.

#### **Node::getDefaultLCID Method**

The **Node** class now provides the **getDefaultLCID** method, which has the following signature.

```cpp
getDefaultLCID(): Integer;
```

This method returns the number of the locale with which the node process was initiated.
Node::getLCIDFromCharacterSet Method

The Node class now provides the getLCIDFromCharacterSet method, which has the following signature.

```java
getLCIDFromCharacterSet(charset: String): Integer;
```

This method returns the locale id corresponding to the short name for the character set specified in the charset parameter.

TranslatableString::updateCompile Method

The TranslatableString class now provides the updateCompile method, which has the following signature.

```java
updateCompile(source: String;
             errorCode: Integer output;
             errorOffset: Integer output;
             errorLength: Integer output): Boolean updating;
```

This method compiles and updates the existing translatable string.

MergeIterator

JADE now provides the MergeIterator subclass of the Iterator class, which implements methods to support iterating over a merged view of two or more compatible dictionary instances. For details, see Volume 2 of the JADE Encyclopedia of Classes.

The MergeIterator class implements the Iterator class interface, with the addition of singleton Dictionary class access methods; for example, getAtKey.

When iterating multiple dictionaries, the merged iterator returns entries in key sequence.

Monitor Enhancements

This section summarizes the JADE Monitor enhancements in this release.

Displayed Data

The JADE Monitor now displays:

- **Lock Chronology** view, which displays acquired locks in chronological order. This can be useful, for example, when diagnosing deadlocks.
- **DbFile Analysis** view now provides the Partition Display group box at the top of the view, and contains the All, Resident, Open (the default value), and Suppress option buttons.
- **Timers** view in the Process Information activity group, which displays information about current timer activity. For details, see “Getting Timer Information” under “Getting Process Information”, in Chapter 4 of the JADE Object Manager Guide.
Monitor Application

In JADE 6.3, the applications JadeMonitorV2 and JadeMonitor are now synonymous. Existing shortcuts to the old name (JadeMonitor) and the new name JadeMonitorV2 continue to work; that is, to launch the JadeMonitorV2 application.

At run time, the application name that is displayed in the Users view indicates the monitor application name that was invoked, although identical code is executed by both the old and the new names.

Security

JADE now provides security hooks to enable you to tailor access to the JADE Monitor. For details, see “JADE Monitor Security”, in Chapter 2 of the JADE Object Manager Guide.

Object Cache Priority

Setting the cache priority to zero (0) does a more complete job of removing objects from cache, by including subobjects.

In earlier releases, String Large Objects (slobs) and Binary Large Objects (blobs) were not removed from cache as the result of setting cache priority to zero (0) and when a collection header was specified, collection blocks were not removed from cache.

From this release, slobs, blobs, and collection blocks are removed from cache as a result of the cache priority of the owning object being set to zero (0).

Patch Control

This section summarizes the patch control enhancements in this release.

Checking Methods Out to Multiple Deltas

From this release, you can check out a method to multiple deltas, which is the default action.

The JADE initialization file now provides the [DeltaOptions] section, which contains the SingleDelta parameter that is set to false by default and which is read when the JADE development environment is next initialized on that node. If you want to retain the functionality of earlier JADE releases so that a method can be checked out to one delta only, set this parameter to true.

Setting the parameter to true is ignored for methods that have already been checked out to multiple deltas; it is effective only for all methods that are subsequently checked out.

Comparing Method Sources

The Compare Sources window now provides combo boxes that enable you to select the display of the source that is currently checked out and the method source in the delta that is currently set. You can now also view the original source (that is, before the method was checked out to a delta).

If a method has been checked out to more than one delta, the combo boxes enable you to select the delta in which you want to compare the method source.
Extracting Patch Number Changes in a Specific Delta

The Extract By Patch Number dialog, accessed from the Extract command from the Patches menu, now provides the Delta combo box, which enables you to select the delta whose patch number changes you want to extract.

If a delta is currently set, patch number changes are extracted from that delta by default. If no delta is set, the un-checked out versions of the methods are extracted.

You can now also specify the delta into which loaded methods are checked out. For details, see “Loading Schemas”, earlier in this document.

Extracting and Loading a Patch History

You can now use the jadclient non-GUI client application to automate the extraction and loading of a patch history, passing command line arguments after the endJade parameter to specify your extract and load requirements, using the respective JadeExtractPatchHistory and JadeLoadPatchHistory applications in the JadeSchema schema.

For details, see “Extracting and Loading a Patch History”, in Chapter 3 of the JADE Development Environment Administration Guide.

JadePatchControlInterface::getCheckedOutEntitiesForPatch Method

The JadePatchControlInterface class now provides the getCheckedOutEntitiesForPatch method, which returns checked out methods for a specified patch number and delta.

Merging Compared Source

The Compare Sources window now provides a merge editor pane below the original source code and updated source code panes.

The popup menu accessed from either of the source code panes provides the Copy to Merge Window command, which copies all of the code in that pane to the merge editor pane. You can then change the copied source and then save it by clicking the Save button. The saved source is saved to your current delta.

Setting a Delta

Forms are no longer closed when you set or unset a delta. The method source and text windows, if applicable, are updated to reflect the change in delta status. (See also “Specifying a Delta on the Command Line of the jadclient Program”, later in this document.)

Setting a Patch Number

The File menu in the JADE development environment now provides the Set Patch Number command, which displays the Set Patch Number dialog that enables you to select the patch number that you want to use in the current schema.

All recently used patch numbers in the current schema are available for selection.
Specifying a Delta on the Command Line of the jadclient Program

When running a non-GUI client application using the jadclient executable, you can now specify the optional delta command line argument, as shown in the following example.

```
jadclient schema=ASchema app=NonGUIApp ini=c:\Jade\test\jade.ini path=c:\Jade\system delta=foxtrot
```

Printing

The CMDFont and CMDPrint classes now provide the printerDC64 property, which contains the 64-bit Windows device context of the printer.

Process Class

The Process class now provides the property and method summarized in the following subsections. For details, see Volume 2 of the JADE Encyclopaedia of Classes.

`countQueuedNotifications` Method

The countQueuedNotifications method returns the number of unprocessed notifications queued for the calling process.

You can call this method only on the process instance of the current process. An exception is raised if you call it on an instance of another process.

`number` Property

The number property is a read-only Integer attribute that is an internal number, relative to the system, that distinguishes the process from other concurrent processes.

The value is zero (0) until the process has successfully passed the validation and initialization stages.

Prohibiting Persistent Transactions and Updates in the Current Process

The Process class prohibitBeginTransaction and prohibitPersistentUpdates methods now prohibit actions as control changes from client method to server method, and the reverse. Process state is synchronized across nodes, where required.

In addition, these methods now return a Boolean value (that is, they return true if the persistent transaction or update was prohibited; otherwise they return false).

Real Primitive Type

This section summarizes the new methods and constant in the Real primitive type.

`arcTan2` Method

The arcTan2 method which returns the arc tangent (inverse tangent) of the specified parameter divided by the receiver. The resulting value represents an angle in degrees.
Determining or Assigning a Floating Point

The Real primitive type is implemented using floating point values. Floating point numbers can have special values; for example, infinity or NaN (Not a Number). Some floating-point calculations can produce infinity or NaN as the result of an operation on invalid input operands.

The following methods are therefore now specified in the Real primitive type.

- The getFloatingPointClassification method enables you to determine the type of floating point of a Real value.
- The setFloatingPointClassification method enables you to explicitly assign a floating point value to a Real value, by specifying one of the following Real primitive type constants in the classification parameter of the method:
  - FP_Classification_NotANumber
  - FP_Classification_PosInfinity
  - FP_Classification_NegInfinity

Note that there are many possible NaN representations. JADE returns a single NaN representation. A NaN does not compare equal to any floating point number or NaN, even if the latter has an identical representation.

Real Primitive Type Constants

The Real primitive type now provides the following constants.

- FP_Classification_NotANumber
- FP_Classification_PosInfinity
- FP_Classification_NegInfinity
- FP_Classification_Zero
- FP_Classification_SubNormal
- FP_Classification_Normal

Relational Population Service (RPS)

The following subsections summarize the RPS changes in this release.

Configuring the RPS Node on a Secondary RPS System

The Configure RPS Node dialog, documented under “Configuring your RPS Node”, in Chapter 2 of the JADE Synchronized Database Service (SDS) Administration Guide”, now provides the Database Type that enables you to select the type of SQL Server, the Use Sqlcmd check box and corresponding Server Name text box that enable you to specify that the sqlcmd utility is used to generate SQL scripts, and the CodePage combo box, which enables you to specify the code page to be used for loading extracted data.
Requirements for RPS Nodes

With the Relational Population Service (RPS) that provides automatic replication of objects from a JADE database to one or more relational databases when running an RPS node, the following SQL Server ODBC drivers must be used for the SQL Server versions.

- Microsoft SQL Server 2008 requires SQL Server Native Client 10.0 or later
- Microsoft SQL Server 2005 requires SQL Native Client or later
- Microsoft SQL Server 2000 requires SQL Native Client or later (recommended) or SQL Server

The edition must match the JADE server edition (32-bit or 64-bit) running on the RPS node.

The sqlcmd SQL Server utility must be installed on the RPS node to use the default (recommended) sqlcmd to execute SQL scripts. (For details, see step 6.e under “Configuring your RPS Node”, in Chapter 2 of the JADE Synchronized Database Service (SDS) Administration Guide.)

SQL Server 2005 and 2008 Default Mappings

When the database type of SqlServer 2005 or SqlServer 2008 is selected, the following applies.

- SQL Server 2005 and subsequent SQL Server versions default mappings, as follows.
  - JADE unbounded strings or strings with a length greater than 8000 to VARCHAR(max)
  - JADE unbounded binaries or binaries with a length greater than 8000 to VARBINARY(max)
- SQL Server 2008 and subsequent SQL Server versions default mappings, as follows.
  - JADE Date to DATE
  - JADE Time to TIME
  - JADE TimeStamp to DATETIME2

Changing the database type can create update scripts that will be executed during the reorganization replay on the RPS node.

RelationalView Class

The RelationalView class now provides the methods summarized in the following table.

<table>
<thead>
<tr>
<th>Method</th>
<th>Returns true if the receiver is being used as an…</th>
</tr>
</thead>
<tbody>
<tr>
<td>isODBCRelationalView</td>
<td>ODBC relational view</td>
</tr>
<tr>
<td>isRpsMapping</td>
<td>RPS mapping</td>
</tr>
</tbody>
</table>

For details, see Volume 2 of the JADE Encyclopaedia of Classes.

Reorganization

This release provides:

- Faster collection population for specific schema changes
- Options to reduce disk space requirements
Note  Fast building of collections is disabled by default. It must specifically be enabled by setting the FastBuildBTreeCollections parameter in [JadeReorg] section of the JADE initialization file to true.

When the FastBuildBTreeCollections parameter in [JadeReorg] section of the JADE initialization file is set to true, the building or rebuilding of some ObjectSets and MemberKeyDictionaries is moved from the relationship maintenance phase to the object conversion phase of the reorganization. This significantly reduces the elapsed time of large database reorganizations involving collection maintenance, by using a faster extract, sort, and build algorithm and by allowing the collection maintenance to be performed in parallel by multiple reorganization workers using much less random I/O activity.

Caution  The resource cost for this feature is the additional disk space required for the extract and sort files (up to three times the total size of the collections being built or rebuilt).

The term ObjectSet includes both subclasses of the Collection class ObjectSet and subclasses of the Collection class Set whose member type is an object. MemberKeyDictionaries with key paths and collection references with constraint methods are supported.

Collection fast-build is invoked when the collections of a MemberKeyDictionary or ObjectSet reference are built or rebuilt and the inverse is an existing object reference; that is, a many-to-one relationship. The update mode of the collection reference must be automatic and that of the object reference must be manual.

In this release, this feature is limited to non-updating reorganizations.

If you enable fast-build functionality, you must also disallow updates when you initiate the reorganization using the:

- Allow updates check box on the Classes Needing Reorg dialog in the JADE development environment must be unchecked.
- reorgAllowUpdates command in the jadload or jadloadb command line must be set to false.
- reorgAllowUpdates parameter in the JadeReorgApp application non-GUI jadclient command line must be set to false.

For details, see Chapter 19, “Database Reorganization”, in the JADE Developer’s Reference. See also “Reorganization Parameters” under “JADE Initialization File”, earlier in this document.

Replication Framework

The Replication Framework replicates selective changes between participating JADE systems; for example, physically different but logically similar JADE systems. Although the JADE systems can be identical, they do not have to be. If they are not identical, you must map the classes and properties involved on each side of the replication.

The framework includes object synchronization and conflict resolution handing.

This enables you to have JADE environments where one side is a full JADE system (the server) and the other (the client) has a limited functionality, typically mobile, JADE system on a device such as a PDA. The client runs an application in a separate schema from the server, with a limited number of classes that have only the properties required for the application.

The Replication Framework enables initial bulk-loading to the client, subsequent replication of server changes to the client, replication of client changes to the server, and requests for data from the client.
Replication can be one-sided, where updates are expected on the server only (that is, the client is read-only) or two-way, where both server and client can update objects. It also includes the checking in and out of objects.

The Replication Framework provides methods that enable you to perform application-specific processing and to trace transactions.

The Replication Framework is implemented as the ReplicationPackage JADE package in the JadeReplicationSchema schema. This schema is released as a separate encrypted source schema and is not automatically loaded when you install JADE.

For details, see the JADE Replication Framework User’s Guide.

Security

For security reasons, the operation of a JADE password text box (the TextBox class passwordField property) has been changed so that an attempt to copy (using the CTRL+C shortcut keys) or cut (using the CTRL+X shortcut keys) the password text into the clipboard is now rejected with a message beep. Pasting into a password field continues to be permitted.

See also “Monitor Enhancements”, earlier in this document.

TimeStampOffset Primitive Type

JADE now provides the TimeStampOffset primitive type, which represents a date and time value, together with an offset that indicates how much that value differs from Coordinated Universal Time (UTC). For details, see the JADE Encyclopaedia of Primitive Types.

Use this primitive type in applications that:

- Uniquely identify a single point in time. You can use the TimeStampOffset primitive type to unambiguously define the meaning of the current time, to log transaction times, to log the times of system or application events, and to record file creation and modification times.
- Perform general date and time arithmetic.
- Preserve multiple related times, as long as those times are stored as two separate values or as two members of a structure.

A TimeStampOffset local variable is initialized with the current date, time, and offset from UTC each time the method in which it is defined is invoked.

The initial value is relative to the presentation client, if JADE is running in thin client mode.

Note: External functions do not support TimeStampOffset as parameters and the return type.

For details about converting from and to a TimeStampOffset primitive type, see “Converting Primitive Types”, in Chapter 1 of the JADE Developer’s Reference.
The following table summarizes the timestamp offset methods defined in the `Application` class. (For details, see Volume 1 of the JADE Encyclopaedia of Classes.)

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>actualTimeStampOffset</code></td>
<td>Returns the current date, time, and UTC offset as a <code>TimeStampOffset</code> value</td>
</tr>
<tr>
<td><code>actualTimeStampOffsetAppServer</code></td>
<td>Returns the current date, time, and UTC offset of the application server as a <code>TimeStampOffset</code> value</td>
</tr>
<tr>
<td><code>actualTimeStampOffsetServer</code></td>
<td>Returns the current date, time, and UTC offset of the database server as a <code>TimeStampOffset</code> value</td>
</tr>
</tbody>
</table>

**Transient Database File Naming**

Since the JADE 6.2 release, transient database overflow file names allow the associated process to be readily determined. (For details about the file naming conventions of this file, see “Transient Database File Analysis”, in Chapter 3 of the JADE Database Administration Guide.)

The file name now includes a node number; that is, each transient database file is named using the following convention.

```
tdb_<host-name>_<pid>_<node-number><designator>.tmp
```

The `<node-number>` value, which is reserved for future use, is zero (0) in this release.

**Translating JadeRichText Context Menus**

The `JadeRichTextFindReplace`, `JadeRichTextInsertTable`, `JadeRichTextParagraph`, and `JadeRichTextPopupMenu` subclasses of the `Form` class now handle the translation of form, menu, and control captions.

To translate these forms, you must re-implement the `app.getRootSchemaFormTranslation` method in your user schema and supply the translation of the required entities.

**Web-Enabled Forms**

The physical directory for Web-enabled forms, if specified, is no longer validated or created automatically.

To create the physical directory, you must now click the `Create` button in the Image Directory Mapping group box of the JADE Forms folder on the Define Application dialog Web Options sheet. This then validates the directory specified in the Physical Directory text box and creates it.

**Web Services Extensions**

JADE Web services now provide:

- A Rich Internet Applications (RIA) framework, summarized in the following subsection.
- Support for packages and interfaces (the WSDL generator looks for imported classes that are being used in the Web service method signatures, exposes these classes along with the properties that have been imported for these classes, and displays them for selection in the WSDL generate wizard).
- Multiple exposures for each Web service application.
Ability to programmatically:

- Create a Web service application, by using the `Schema` class `createWebServiceApplication` method.
- Generate a WSDL, by using the `Schema` class `generateWSDL` method.
- Add, remove, and obtain the HTTP authorization header field to a Web service consumer request, by using the `JadeWebServiceConsumer` class `addHttpHeader` and `getHttpHeader` methods.
- Use the `JadeWebServiceProvider` class `getServerVariable` method to return server-based information; for example, return the IP address by using the following method call in your Web service provider code.
  ```javascript
  self.getServerVariable('REMOTE_ADDR');
  ```

- The ability to control the amount of text to be displayed in the JADE Web Application Monitor window by using the new `output_maximum_length` option in the XML-based Web configuration file.
  For details, see the **JADE Web Application Guide**.

**Rich Internet Applications (RIA)**

The standalone Rich Internet Applications (RIA) framework creates a small set of `javascript` files that can communicate with a specified JADE or external Web service.

The framework makes it easier for you to invoke JADE Web services from JavaScript clients and to process the data that is returned from the Web services, by removing much of the underlying “plumbing” code that currently needs to be written.

For details, see Chapter 4, “Using the JADE Rich Internet Application (RIA) Framework”, in the **JADE Web Application Guide**.

**XMI Support**


The JADE development environment File menu now provides the `XMI Import` command, which accesses the Extract Schema dialog, to enable you to select the name and location of the XMI file to extract and the location of the output files.

For details, see “XML Metadata Interchange (XMI) Support”, in Chapter 26, in the **JADE Development Environment User’s Guide**.