Upgrading to the JADE 2016 Release

VERSION 2016.0.02

jade

Contents

Upgrading to the JADE 2016 Release

This document covers the following topics.

Tip For details about using a web browser to view the JADE product information, see "JADE HTML5 Online Help", in Chapter 2 of the *JADE Development Environment User's Guide*. For details about using Acrobat Reader to view the JADE product information, 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**) 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** document in the **documentation** directory.

JADE Release Support

For details about the:

- JADE release policy, see https://www.jadeworld.com/pdf/tech/JADE_ReleasePolicy.pdf.
- JADE release schedule, see https://www.jadeworld.com/developer-center/download-jade/release-schedule.

JADE 2016 is built using Microsoft Visual Studio 2013, which requires the installation of appropriate C++ runtime binaries.

For details about the deimplementations and deprecations in this release, see the following subsection.

Deimplementations and Deprecations

This section contains the deimplementations and deprecations in this release. (See also "JadeHTMLClass::buildFormActionOnly Method (PAR 62632)", under "Changes and New Features in JADE Release 2016", later in this document.)

Binary Type pos Method Deprecation

The **Binary** primitive type **pos** method has been deprecated in this release, and will be deleted in a future JADE release.

Note Use the **Binary** type **posBinary** or **posByte** method instead of the deprecated **pos** method.

Compact JADE

As notified in JADE 7.1.08 and later, Compact JADE is no longer available in this product release.

Note Compact JADE in thin client mode will continue to be available for the lifetime of JADE 7.1.

Customizing the Presentation Client Installer

As notified in JADE 7.1.09, the Customizable Presentation Client Installer (CPCI) has been deprecated in JADE 2016.0.02.

Highlights in this Release

The highlights in JADE release 2016, which help you to deliver high-performance, interoperable applications on Windows for both 32-bit and 64-bit platforms, are as follows.

- A large number of changes and enhancements to the development environment to improve the developer experience, including:
  - Refreshed look and feel, with a more-modern, flatter appearance
  - An editor clipboard toolbar
  - Locating unused classes, properties, class constants, and methods in one or more schemas
  - Script management, to facilitate script reuse and to reduce duplication
  - JADE Debugger and Painter enhancements
  - JADE Inspector enhancements

  For details, see "JADE Development Environment" under "Changes and New Features in the JADE 2016 Release", later in this document.

- Debugging Lock Exceptions

  JADE now supports the optional recording of the current call stack when a process locks an object. Any process can retrieve this information while the lock is held; for example, you can use it to help find and resolve locking problems during application development, by tracking down where in the code any long-lived lock was obtained. This information, which is passed to the lock manager and stored in the lock entry, can be retrieved by any process while the lock is held.

  For details, see "Debugging Lock Exceptions (NFS 63339)", later in this document.

- Standalone JavaScript Object Notation (JSON) functionality

  JADE now provides JSON functionality that is independent of the Representational State Transfer (REST) Application Programming Interface (API), which enables you to create, load, unload, and parse JSON in the same way you can with XML.

  For details, see "Standalone JSON Functionality (NFS 57762)", later in this document.

- Type methods

  JADE now supports *type methods*, which provide a way of calling a method declared on a type (or class) without having to have an instance of the type. While the method is declared on a type and has the same scope as an instance method, at run time the receiver is the type on which the method is declared; *not* an instance of the type.

  For details, see "Type Methods (NFS 63225)" under "JADE Development Environment", later in this document.

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.

https://www.jadeworld.com/developer-center/jade-support/parsys

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 **2016** list item).
3. Click the **Search** button.

Upgrading to JADE 2016

This section covers the following topics.

Caution Before you upgrade to JADE 2016, refer to "JADE 2016 Changes that May Affect Your Existing Systems", elsewhere in this document.

As with any JADE release (for example, upgrading to a minor release or to a major feature release from an earlier JADE version), you must 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*.)

Upgrading to JADE 2016 from JADE 7.1

If you want to develop your own upgrade process, refer to the JADE install and upgrade steps documented in the **ReadmeInstallSteps.pdf** document in the documentation directory.

Note Example files are not part of the installation and must be downloaded or installed from the JADE web site, if required.

The JADE Setup program enables you to upgrade your binary and database files to JADE 2016 from JADE 7.1, by performing the following actions.

1. On the JADE 7.1 system, carry out the following certify operations. Proceed to the next certify operation only when any and all errors reported in the current operation are resolved.

   a. A physical certify using JADE Database utility (**jdbutil.exe** or **jdbutilb.exe**), to ensure that the system is structurally correct. (For details, see Chapter 1 of the *JADE Database Administration Guide*.)

   b. A meta logical certify, to ensure that the meta model is clean. (For details, see "Running a Non-GUI JADE Logical Certifier", in Chapter 5 of the *JADE Object Manager Guide*.)

   c. A logical certify, to ensure that the user data is referentially correct. (For details, see "Running the Diagnostic Tool", in Chapter 5 of the *JADE Object Manager Guide*.)

   Note If you are unsure how to interpret the information output by the certify process, first refer to "Logical Certifier Errors and Repairs", in Chapter 5 of the *JADE Object Manager Guide*, and if you are still unsure, contact JADE Support (jadesupport@jadeworld.com) for advice.

2. Use the JADE Database utility to take a full backup of your existing JADE 7.1 database.

   Caution If the upgrade should fail, you will need to restore this backup and then retry the upgrade process when all of the conditions that caused the failure have been addressed.

3. Installing the JADE ODBC drivers and the Microsoft Visual C++ redistributable packages requires administrator rights, so ensure that you have the appropriate privileges.

4. Run the JADE 2016 installer, available from https://www.jadeworld.com/developer-center/download-jade.

Note The **Custom** type applies only to a **Fresh Copy** installation type, and is not relevant when upgrading.

The **SDS/RPS Database Server** option applies only to 64-bit **Feature Upgrade** installation type.

5. A warning message may be displayed if the upgrade validation process has not completed. If so, check the **jadeupgrade.log** file for information about what needs to be modified in your user schemas to pass the validation and enable application execution.

   If the validation needs to be run again, see the **ReadmeInstallSteps.pdf** file in the documentation directory for instructions.

6. 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. The **ReadMe.txt** file contains late-breaking important information not possible to publish in this document.

7. Use the JADE Database utility to take a full backup of your JADE 2016 database.

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 **jommsg*n*.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.

To perform the validation from the command line, see the **ReadmeInstallSteps.pdf** file in the **\documentation** directory.

Running Two Releases of JADE on the Same Workstation

You can have any number of releases of JADE installed on the same workstation. If ODBC is installed, only the last installation of the JADE ODBC driver is available from the ODBC Data Source Administrator.

JADE Thin Client Upgrade

When upgrading a presentation client to JADE release 2016, ensure that you have the appropriate privileges or capabilities to install applications.

The configuration of User Account Control (UAC) and your current user account privileges may affect the behavior of the upgrade to JADE 2016. For details about UACs, standard user accounts, and administrator accounts, see the Microsoft documentation.

If JADE is installed in the **\Program Files** directory (or **\Program Files (x86)** directory on a 64-bit machine with 32-bit JADE binaries):

- If the 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 an SDS Native or RPS Secondary System

SDS secondary databases can be upgraded. For details about how to do this, see the **ReadmeInstallSteps.pdf** file in the **\documentation** directory.

JADE 2016 Changes that May Affect Your Existing Systems

This section describes only the changes in the JADE 2016 release that may affect your existing systems. Some changes may result in compile errors during the load process, or cause your JADE release 2016 systems to behave differently.

Form Class isModal Method (NFS 53547)

The **Form** class now provides the **isModal** method, which returns **true** if the form was displayed using the **Form** class **showModal** method, or it returns **false** if the form has not been displayed or it was displayed using the **Form** class **show** method.

Imported Control Class Event Truncation (PAR 63460)

Although the maximum size of identifier names was increased to 100 characters in JADE 7.1, the .NET and ActiveX import process restricted the length of imported control events names to 14 characters. When a control is added to a form, event method names became *control-name_event-name*, so if the event name was long and not truncated, to construct a valid event method name meant that the control name had to be unacceptably short.

From this release, imported control event names are restricted and truncated to 49 characters, which allows a control name to be up to 50 characters.

Note Reloading an assembly that uses the old naming schema retains the previously assigned names so that your existing logic is not affected. As a result, you must delete the assembly and reload it, to get the new naming convention.

List Box Icons

With the JADE development environment having been refreshed, its more-modern, flatter appearance has resulted in list box icons (for example, to expand and collapse nodes) looking different.

If you have a form with a number of list box controls and user-defined picture images, check to make sure that the layout has not been compromised.

Note This change affects only the *default* list box icons. If you have specified your own icons, these are not affected.

OpenSSL Library Implementation (PAR 63548)

From JADE 7.1.07, usage of the **JadeSSLContext** class **MethodSSLv2**, **MethodSSLv23**, or **MethodSSLv3** constant was overridden at runtime with the new default **MethodTLSv1_2** constant, and a message was written to the **jommsg log**. This may have affected usages of the **JadeHTTPConnection** class.

As the **MethodSSLv2**, **MethodSSLv23**, or **MethodSSLv3** constants have been removed in JADE 2016, update your source code if you used any one of these class constants in JADE 7.1, as the upgrade validate step will fail if you are still using any one of these deimplemented constants.

See also "Secure Sockets Library (PAR 63548, NFS 63550)" under "Changes and New Features in JADE Release 2016.0.01", later in this document.

Web Session Timeout (PAR 63011)

In earlier releases, if a user schema re-implemented the **timerEvent** method on a **WebSession** class being used by a Web Service Provider application, JADE did not start the **WebSession** time-out timer. This caused web sessions not to time out.

This behavior has been changed so that if the web session **timeout** value is set (in the **ReadTimeout** parameter of the [WebOptions] section of the JADE initialization file, by XML configuration, or in your application logic), when a web session is created, the time-out timer will be started regardless of whether or not the Web Service Provider application in your user schema has re-implemented the **timerEvent** method on the **WebSession** subclass being used.

Note To enable the session time out process to function correctly, it is then the responsibility of that re-implemented method to call the **inheritMethod** instruction.

Changes in JADE Release 2016.0.02

This section contains details about product and documentation changes in JADE release 2016.0.02. For details about release 2016.0.01 (the first general release of JADE 2016), see "JADE 2016 Changes that May Affect Your Existing Systems" and "Changes and New Features in JADE Release 2016.0.01", elsewhere in this document.

Additional Command Line Handling for Applications Run using jade.exe (NFS 64920)

When initiating a JADE client application using the JADE executable (**jade.exe**), JADE now provides the ability to collect strings from the command line and pass them to the application's initialize method.

Note This feature is available for any application type run from the client, including a standard (fat) client, a thin (presentation) client, non-GUI applications, and so on.

Only the application initiated using the command line parameters uses these command line arguments.

The implementation is as follows.

1. The new arguments recognized in the command parameters are **StartAppParameters** and **EndAppParameters**.

2. The elements between the **StartAppParameters** and **EndAppParameters** arguments are added to a transient **HugeStringArray** object that is created.

   The added elements are a unary string (for example, **report**) or a coupled pair separated by a **=** character (for example, **option=true**). For example, **StartAppParameters fileName = "g:\temp\manifest today.txt" report output=true EndAppParameters** results in a **HugeStringArray** of:

   a. fileName=g:\temp\manifest today .txt

   b. The test name

   c. output=true

3. The built **HugeStringArray** object is passed to the defined **initialize** method for the invoked application. That method must therefore expect a parameter of a **HugeStringArray** type. If not, exception 4027 *(Method called with incorrect number of parameters)* is generated for the parameter mismatch.

When processing the elements after the **StartAppParameters** argument:

- Any double or single quotes around elements are dropped.
- Any spaces just before and directly after the **=** sign are dropped.
- The application parameters begin with the first element after the **StartAppParameters** argument and are ended by the **EndAppParameters** argument, the **endJade** argument, or by the end of the command line (whichever comes first).
- The elements are separated by spaces.
- Leading and trailing spaces in each string element are removed.

Note also:

- If no such command line parameters are specified, the **initialize** method is passed a null value if it requires a parameter.
- The created transient object is deleted when the **initialize** method call completes.
- This feature applies only to the application being initiated from the command line of **jade.exe**.

This feature was implemented to allow **RootSchema** non-GUI applications to run using a **jade exe** client instead of the non-GUI client **jadclient** executable, but it is also available for user application use. The first such **RootSchema** applications that you can run using this mechanism are the **JadeUnitTestBatch** and the new **JadeUnitTestGuiNoForms** applications. (In earlier releases, you could run the **JadeUnitTestBatch** application only as a standard client using the **jadclient** executable.)

For an example of the **jade.exe** command line for a non-GUI client application, see "Running Unit Tests as a Fat or a Thin Client in Batch Mode (PAR 65202, NFS 64920)" and "Running Unit Tests that Require GUI Components in Batch Mode (NFS 65222)", later in this document.

Note Although **JadeUnitTestBatch** and **JadeUnitTestGuiNoForms** applications are the only **RootSchema** applications that can currently make use of this feature, other applications will be able to do so in future releases.

Call Stack Browser (NFS 64426)

The Call Stack Browser (invoked from the default exception handler when an exception is debugged) and the JADE Debugger Call Stack window now support copying the current call stack to the clipboard as a string, by:

1. Right-clicking on the call stack results.
2. Selecting the **Copy Call Stack to Clipboard** command from the popup menu that is then displayed.

Displaying a Check Box in a Disabled Cell (NFS 64834)

In earlier releases, check boxes were not shown as **disabled** in cells whose input was disabled.

A disabled cell with **InputType_CheckBox** now draws the check box as disabled.

Docking Control Sizing (PAR 65245)

In earlier releases, the maximum and minimum sizes of JADE docking controls could be ignored if you had:

- One control only aligned horizontally or vertically in the container
- All of the dock controls in the container had the maximum or minimum sizes set, or both the maximum and minimum sizes set

When JADE docking controls were aligned horizontally or vertically in a parent container, they were resized to entirely fill the client area of their parent. The implementation of the **JadeDockBase** class maximum and minimum height and width properties introduced a conflict of behavior. JADE resized the controls to entirely fill the client area of their parent, so that when all of the child controls were JADE dock controls and they all had maximum or minimum sizes set, these sizes could be ignored to accomplish the filling of the client area.

This behavior has been changed so that the minimum and maximum sizes are now always honored. This can mean that the client area of the parent may not be filled or the controls may not all fit the client area if all of the children have the horizontal and vertical maximum and minimum sizes set.

If at least one child does *not* have a maximum or minimum size set, the filling of the client area of the parent still occurs.

Drawing on the Desktop (PAR 65731)

The latest version of the Windows 10 operating system no longer properly handles the dragging of a drawing on the screen; for example, when drawing a dragging rectangle for a dock toolbar control that is dragged.

This mechanism has now been rewritten to use newer Windows facilities to achieve the required drawing.

Window Class drawDeskTopRectangle Method

The existing **Window** class **drawDeskTopRectangle** method has been changed to work as required using newer Window facilities. This method now creates a transparent desktop window onto which the rectangle is drawn. Any white pixels are treated as being transparent.

This window is used for any subsequent calls to the **drawDeskTopRectangle** method made on the associated window and is repositioned and resized each time.

Each call to the method erases any previous drawing and draws the new rectangle pattern.

Note The drawing, however, no longer uses the Xor operator to combine a draw pattern pixel with a background pixel on the desktop.

The window is destroyed if the effective rectangle passed in the call is the same as the previously drawn rectangle or the window on which the call was made is destroyed. This means that existing logic that redraws the previous rectangle drawn (double xor) will work correctly. However, the drawing will flash as the window is repeatedly created and destroyed.

To improve the user experience, change the logic to remove the second draw call to erase the rectangle, and redraw it only when the window is no longer required.

The window is also destroyed if a call is made with **-1** as the first parameter (**borderStyle**).

Note Any window can have an associated drawing, thus allowing multiple rectangle patterns to be visible at once. Each drawing is independent of the other.

The **borderStyle** first parameter of the method call (formerly the **pattern** parameter, in earlier releases) can now be one of the following values.

Parameter Value	Description
-1	Destroy drawing window
0	Hatch style 45 degrees left to right
1	Cross-hatch
2	45 degree cross-hatch
3	Hatch style 45 degrees right to left
4	Horizontal
5	Vertical
6	Half tone
7	Solid

Any other value is treated as no drawing required.

Note The **drawDeskTopRectangle** method does *not* draw the inside of the rectangle and the rectangle border is not drawn if the value of the **borderWidth** parameter is less than or equal to zero (**0**). The documentation in earlier releases was therefore incorrect.

The **drawDeskTopRectangle** method signature has been altered to be more correct, as follows.

```
drawDeskTopRectangle(borderStyle: Integer;
                     x1:          Integer;
                     y1:          Integer;
                     x2:          Integer;
                     y2:          Integer;
                     borderColor: Integer;
                     borderWidth: Integer);
```

Window Class drawDeskTopRectangle Method

From this release, JADE provides the **Window** class **drawDeskTopRectangleEx** method, which has the following signature.

```
drawDeskTopRectangleEx(borderStyle: Integer;
                       x1:          Integer;
                       y1:          Integer;
                       x2:          Integer;
                       y2:          Integer;
                       borderColor: Integer;
                       borderWidth: Integer;
                       innerStyle:  Integer;
                       innerColor:  Integer);
```

This method does the same as the **drawDeskTopRectangle** method except the:

- Inner part of the rectangle is drawn using the value of the **innerStyle** parameter (with the same pattern options as those for the **drawDeskTopRectangle** method **borderStyle** parameter listed earlier in this topic) and the color specified in the **innerColor** parameter unless the value of the **innerStyle** parameter is **-1**.
- Drawing window is not destroyed if the same rectangle is drawn a second time.
- Drawing window is destroyed only if both the **borderStyle** and **innerStyle** parameters are both **-1** or the associated window is destroyed.

Event Mapping (NFS 65566)

The **setEventMapping** method defined in the **MenuItem** and **Window** classes are documented as being expensive when executed at run time.

These implementations have now been changed to significantly improve performance by caching the mapping on each JADE node. The first call for a specific mapping performs the same signature check and is still expensive.

Repeat calls for a mapping that has been previously used is recognized and the signature check is not repeated unless the timestamp of the mapped method has changed since the previous signature check. The cost of reloading a form that assigns event mappings is therefore subsequently less expensive on that node. If an application server is involved, only the first assignment by any user performs the signature check. Subsequent repeat calls for any user on that application server avoid that overhead.

In addition, the **MenuItem** class and the **Window** class now provide the following methods, which allow you to set event method mappings.

```
setEventMappingEx(eventMethod: Method; mappedMethod: Method);
```

The event method being mapped and the mapped method are passed as parameters. The underlying logic, therefore, does not have to find the methods by name, making the execution more efficient; for example:

```
mnuCustomer.setEventMethodEx(MenuItem::click, CustomerDialog::customerMenu);

btnAction.setEventMethodEx(Button::click, MyForm::myOtherMenu);
```

The **eventMethod** parameter specifies the event method, which must belong to the class of the receiver of the **setEventMappingEx** call.

The **mappedMethod** parameter is the method to which the event is mapped, and it must belong to the form of the receiver.

Tip These new methods are equivalent to the original **setEventMapping** methods but as they are more efficient, you should use them to improve performance.

Executing Type Methods (PAR 65154)

In the earlier 2016 release, you were unable to execute a type method from the non-GUI client application.

The **jadclient** non-GUI client application now supports the **executeTypeMethod** command line parameter, which indicates that the method specified in the **executeMethod** parameter is a type method. The values for the **executeTypeMethod** parameter are **false** (the method is not a type method) or **true** (the method is a type method). The default value is **false**.

File Handling (PAR 65585)

Windows has a maximum path length of 260 bytes.

An operation using a file name with a length that exceeds the Windows maximum path is logged by JADE, identifying the length, the path, and the operation.

To disable this logging, set the value of the **ExtendedLengthPathMessages** parameter in the [JadeLog] section of the JADE initialization file to **false**.

If the **ExtendedLengthPaths** parameter in the [JadeEnvironment] section of the JADE initialization file is set to **true**, the operation is attempted; otherwise error 5029 (*File name not valid for file system*) is raised.

Forced Unit Access (PAR 65128)

In earlier releases, if the write journal drive was an Advanced Technology Attachment (ATA)-type device, you could not rely on data being stable on the media for the **WRITE** command with Forced Unit Access (FUA). Although it has been the standard since ATA-7 (2005), it is not always supported by commodity drives or device drivers.

From JADE releases 7.1.09 and 16.0.02, writes to the audit files are now synchronized on such devices.

HTTP Connections (PAR 65118, PAR 65124)

In earlier releases, the **JadeHTTPConnection** class documentation stated that only the WinHTTP library could be used for a server or in a service. This is incorrect, as WinHTTP or WinINET can be used.

In addition, the documentation for the **EnableWinHTTP** and **EnableWinINET** parameters in the [JadeEnvironment] section of the JADE initialization file stated that the parameters were read when the first JADE Web service consumer is created. This is incorrect, as they are both read when the node is started.

Inserting Schemas with Subschemas (NFS 65331)

In earlier releases, you could not insert a schema with existing subschemas.

The **JadeInsertSchema** application now supports inserting a schema with existing subschemas to a new position in the hierarchy. The parameters for the **JadeInsertSchema** application have been changed so that rather than a single, comma-separated parameter after the **endJade** parameter in the **jadclient** two new parameters are now required to specify the name of the new superschema being inserted and the name of the above which it is to be inserted.

The parameters passed to the **JadeInsertSchema** application are **superschema=*superschema-name*** and **subschema=*subschema-name***; for example, if the superschema is **TestSchema** and you want to insert it above **CommonSchema**, the **jadclient** command line is as follows.

```
jadclient path=c:\jade\system ini=c:\jade\system\jade.ini schema=RootSchema
app=JadeInsertSchema server=singleUser endJade superschema=TestSchema
subschema=CommonSchema
```

JADE Database

This section describes the JADE database changes in the JADE 2016.0.02 release.

Archival Recovery (PAR 61735)

In earlier releases, all journals except the current journal were released at the end of archival recovery, which impacted on the maintenance of relational view ad hoc indexes.

Following the hostile takeover of a secondary database, any populated ad hoc indexes *may* be left in an inconsistent state with the remainder of the database, as some journals may not have been sent from the primary database and processed before the takeover operation. As existing ad hoc indexes are therefore marked as in error, drop the indexes and then build them, to clear the error state and make them consistent with the state of the rest of the database.

To do this, from the Ad Hoc Index menu in the Ad Hoc Index Browser on the primary database:

1. Select the **Run Ad Hoc Index Controller App** command.
2. Select the index definition that has a red background in the **Status** column, indicating that it is in error.
3. Select the **Drop Index** command.

   A confirmation message box is then displayed stating the number of instances of that index and prompting you to confirm that you do want to drop (unpopulate) it. Click the **Yes** button to confirm that the selected ad hoc index definition is to be dropped.
4. In the Ad Hoc Index Browser table, select the dropped unpopulated index definition that you want to build.
5. Select the **Build Index** command.

Certify Operations (NFS 64695)

The **certify** operation in the JADE database utilities (**jdbutil**, **jdbutilb**, and **jdbadmin**) now prefixes output to the **certify.log** with ***process-id-thread-id***; for example:

```
04230-1850 2017/03/15 13:35:12.043 Certification of file: _userscm [31] commenced
```

JADE Development Environment

This section describes the JADE development environment changes in the JADE 2016.0.02 release.

Bubble Help Display (NFS 65057)

In earlier releases, the display of the bubble help could occur before the **mouseHover** event was executed, and the display may not have been appropriate.

The handling of bubble help has been changed so that it is now aware of whether the window implements the **mouseHover** event. If the window implements the **mouseHover** event, bubble help is displayed after the **mouseHover** event has been executed. This allows the **mouseHover** event to set the **bubbleHelp** text that is appropriate for the mouse position; for example, the list entry that the mouse is over.

Creating a Mapping Method for a Property (NFS 64321)

You can now right-click on a property selected in the Properties List of the Class Browser if you want to automatically generate a mapping method for that property.

The Properties menu now provides the **Create Mapping Method** command, which is enabled if a mapping method can be created for the selected property, in which case the menu command changes to **Create Mapping Method For Property '*property-name*'**. If a mapping method already exists or a mapping method cannot be created for a selected property, the command is disabled.

When you select the command, the Jade Mapping Method Definition dialog is displayed, with the selected property displayed and disabled in the **Name** text box. In addition, the dialog title is displayed as Jade Mapping Method Definition For *class-name*::*property-name*.

You can change the mapping method options and then click the **OK** button. If the method is added, the appropriate mapping method source is constructed. (Alternatively, click the **Cancel** button to abandon the addition of the mapping method.)

Define Class Dialog (PAR 65211)

In earlier releases, the **Text** sheet of the Define Class dialog did not support use of the Enter key, so you could not make multiple-line comments about a class. If a form with a **JadeEditor** control has a default button, pressing the return (Enter) key when the text editor had focus clicked the default button instead of entering the new line character in the text.

The **JadeEditor** control default behavior has been changed so that if it is not read-only, the editor receives the return key when the editor has focus.

Disabling the New Condition Command (PAR 64564)

In earlier releases, when the **Type** sheet had focus in the Methods List of the Class Browser or Primitive Types Browser, the **New Condition** command was enabled in the Methods menu. When this command was selected, a new instance method was created because a type method cannot be a condition method. Focus remained on the **Type** sheet, implying that no method had been added, and the new method was visible only when focus changed to the **All** or **Instance** sheet.

The Methods menu **New Condition** command is now disabled when the **Type** sheet has focus in the Methods List of the Class Browser or Primitive Types Browser.

Editor Pane Resize Bar (PAR 65220)

In earlier releases, if you dragged the resize bar in the editor pane too far upwards, it disappeared completely, and could be restored only using the Ctrl+Shift+W shortcut keys to restore the bar to the bottom of the editor pane.

The top area of a split editor pane now has a minimum height set for the splitter (resize) bar, so the top area can no longer disappear.

Global Search of Checked Out Methods (PAR 64854)

In earlier releases, the global search function ignored methods that could not be modified and unsaved methods that you currently have open.

The global find functionality now searches all methods that meet the specified search criteria, regardless of whether the method cannot be modified or whether it is currently being edited. Any methods currently modified and unsaved by you are also included, but the find action searches only the saved persistent source; not the unsaved source.

Note The search uses the current method source according to whether or not a delta is set. If you have a delta set, the checked-out source of a method is searched if it exists; otherwise the unchecked-out source is used.

The replace functionality remains unchanged and it ignores any methods that cannot be modified, including any unsaved methods that you currently have open.

JADE Painter Properties Dialog Name Control (NFS 64640)

The **Name** combo box control on the JADE Painter Properties dialog now has a style drop-down combo box list (that is, **Style_DropDownComboList**, which enables you to select a control by name by partially entering the required entry so that it is automatically located.

As you enter the text, the first entry that has the specified text as a prefix is selected. Pressing Enter then selects that entry in the combo box, displays the properties for the control in the Properties dialog, and selects the control on the form.

Locks on Changed Methods (PAR 64883)

In earlier releases, when the source of a method was initially changed, the method was locked. However, under certain conditions, the lock was lost; for example, if you opened and closed the same method source in another window or you updated the text data for the method, which meant that another user could also edit the same method at the same time. In addition, the *This source is already locked by you, proceed?* message that was displayed caused confusion, particularly when you attempted to modify the same method in two different windows.

Multiple update actions associated with a method while the source has been modified but not saved now retain the lock until all updated copies of the method source are saved, compiled, or discarded. In addition, if you attempt to edit another copy of the same method in the same JADE environment when the method has already been changed but is unsaved in another window, the following message is displayed in the message box. (The *This source is already locked by you, proceed?* message has been removed.)

```
This method has been already been modified by you and is unsaved in another window.
```

Click:

- **Yes**, to proceed with the change

  The source is changed, and you can continue editing the method. Note, however, that the other changed method source will *not* be updated and you must handle the fact that the source has been changed and is different.

  Note If the method is saved or compiled, the status line of the window with the other unsaved version of the method displays *Method source was changed in another window - the displayed changed source is out of date*.

- **No**, to activate that window and discard the change

  Focus moves to the other window in which the method has been modified.

- **Cancel**, to discard the change

  Focus then returns to the current window.

Method Search over Results of Previous Search (NFS 64771)

In earlier releases, you could not use the results of a previous method search as the starting point for a subsequent search.

The JADE development environment now enables you to perform search and replace actions over the current list of methods displayed on a form, including the list of methods displayed for a class; the list of methods displayed for references of a class, property or method; the list of implementors of a method; and the list of methods displayed after a search.

The Methods menu therefore now includes the **Search/Replace Methods in Method List** command. When you select this command, the Search/Replace the Methods in the Methods List dialog is then displayed. (This is the standard global search and replace form without the controls in which to select the schema and classes.) When you specify your search or replace criteria and then click the **Search** or **Replace** button, only those methods on the form when the search or replace action is performed are searched. For example, you can perform a text search over all methods for a schema using a global search and then perform another search over only those methods found in the first search (and then another search over the results of the second search, and so on).

Opening a Method in a Separate Window (PAR 64558)

In JADE 7.1 and earlier, double-clicking a method in the Methods List of a hierarchy browser resulted in the method source being displayed in a separate editor window, which could then be floated and moved to provide an easy way to view methods side by side, for example.

From JADE 2016, when a method is:

- Double-clicked, it is displayed in a single method source window if the **Reuse Same Method Source Window** check box is checked on the **Method** sheet of the Preferences dialog. (See also "Reusing Forms that Display Method Source (NFS 63234)".) In addition, you could not remove this method from the resulting list; that is, the History List at the top of the single method source window. The single method source window applied only to methods accessed by pressing F11 while editing a method.
- Displayed in the same method source window (for example, by double-clicking a method list box entry, pressing F12 when a method has focus, or pressing F11 when a method is selected in logic), a history of the viewed methods is available. See "Swapping the F11 and F12 Keys in the Editor Pane (NFS 63323)" for details about swapping the actions of the F12 and F11 keys.

As you could not open a method source in a separate window that could be floated unless you first unchecked the **Reuse Same Method Source Window** check box preference, from this 2016.0.02 release, you can now do this by double-clicking a method entry while holding down the Shift key in the:

- Class Hierarchy Browser, Class References Browser, or Senders Browser (search results list), to open a separate method source window for the method regardless of the setting of the reuse option.
- History List of the Reused Method Source form, to open a separate method source window for the method and to remove the method from the History List.

If there are no methods left in the History List, the Reused Method Source form is closed. In addition, the Methods menu **Remove from List** command is now enabled for the Reuse Method Source window when a method entry is selected in the History List. Clicking that command removes the selected method from the History List. If there are no methods left in the History List, the Reused Method Source form is closed.

Process Instance Number Display (NFS 64314)

The Classes In Use Browser and the Process Usages for Class Browser now display the process instance number in addition to the process and any numeric qualifier if multiple JADE processes are running under the same user code, to enable you to more easily identify the processes that need to be closed, for example.

Renaming Methods (NFS 64646)

When a method name was changed in earlier releases, the patch history did not show the changes made under the previous name or names of the method.

The patch history now includes the method history from before the name change (until that rename entry is deleted when the number of patch entries exceeds the defined patch depth).

Notes Only name changes made after JADE 2016.0.02 show the prior history, because existing patch data does not have the required information.

Viewing the history of the new method name includes the history for the old method name but viewing the history of the previous method name does not include the history for the new name.

Searching a Hierarchy Class Browser for an Entity (NFS 63584)

You can now search for a specific property, constant, or method entity in a hierarchy Class Browser, by using one of the following new menu commands.

- Properties menu **Search For Property Name** command (Ctrl+6)
- Constants menu **Search For Constant Name** command (Ctrl+6)
- Methods menu **Search For Method Name** command (Ctrl+7)

Notes To search for a constant name, the **Const** sheet of Properties List in the upper middle pane of the browser must be selected. To search for a property by name, a Properties List sheet other than **Const** must be selected.

This feature is available in the Hierarchy Class Browser only, and not in any other methods browser forms.

When the entity search request is made, the following occurs.

1. A text box is displayed at the top of the list box to be searched.
2. A table is displayed below the text box, and it initially contains all of the elements in the list box to be searched (except duplicate names and class names).
3. Focus is positioned in the text box.

You can perform the following actions.

- Enter text in the text box. After each character is entered, only those entries in the list box that have the specified string somewhere in the entity text (which is case-insensitive) are displayed in the list.
- Press the up or down arrow to move through the list.
- Click an entry in the list to hide the text box and list, and then select the clicked list entry in the original list box.
- Press Enter to hide the text box and list, and then select the currently highlighted entry in the list in the original list box.
- Press Esc to hide the text box and list so that focus is positioned in the original list box without any changes.
- Click another window to hide the text box and list without impacting the original list box.

JADE Inspector Form

The JADE 2016.0.01 release introduced the **History** pane displayed at the right of the Inspector form when the **Use Same Window** command is checked in the Options menu. (This pane contains a hierarchical list box displaying all of the objects that have previously been inspected, indicating the history of how the objects were inspected.)

In JADE 2016.0.02, the **History** pane has been moved to the left of the form.

JADE Logical Certifier Utility

This section describes the JADE Logical Certify utility changes in the JADE 2016.0.02 release.

Meta Data Certification in Multiuser Mode (NFS 63393)

In earlier releases, you could not run the **meta certify** operation in multiuser mode.

Although the JADE Logical Certifier utility must be run in single user mode when certifying user schemas, you can now perform **meta certify** and **repair** operations in multiuser mode.

Note It is not advisable to perform **repair** operations to meta data in multiuser mode.

Meta-Certifying Selected Classes in Batch Mode (PAR 65214)

In earlier releases, the documentation did not state that for a meta-certify operation, you can use the **_metacert.in** file to specify the **RootSchema** classes that are to be certified.

The format of the **_metacert.in** file is the same as that of the **_logcert.in** file except that the **Schema** and **AllSchemas** values are not valid, because the certification of meta data is performed at **RootSchema** level so you cannot filter the schema selection.

JADE Profiler

This section describes the JADE Profiler changes in the JADE 2016.0.02 release.

Method Profiling (NFS 64929)

The JADE Profiler now records calls to both JADE and external methods. The JADE Profiler report output therefore now contains information for *all* method calls, which gives a more-complete picture of activity.

In addition, the default method count has increased from 10 to 100.

Note Method profiling is used by the Method Analysis feature in the JADE Monitor and it can also be called in application code.

It is recommended that when investigating application performance, only one of the JADE Profiler, JADE Monitor, or method profiling is used at any one time, as the results reported when any of these are combined is undefined.

You can specify the default values for the following **JadeProfiler** class properties in the corresponding parameters in the [JadeProfiler] section of the JADE initialization file.

JadeProfiler Class Property	[JadeProfiler] Section Parameter
methodCount	MethodCount=<integer>
reportActualTime	ReportActualTime=<boolean>
reportCacheStatistics	ReportCacheStatistics=<boolean>
reportLoadTime	ReportLoadTime=<boolean>
reportStatistics	ReportStatistics=<boolean>
reportTotalTime	ReportTotalTime=<boolean>
reportMethodSize	ReportMethodSize=<boolean>
profileRemoteExecutions	ProfileRemoteExecutions=<boolean>
reportInCSVFormat	ReportInCSVFormat=<boolean>
fileName	ResultsFile=<filename>

Profiling Output to a CSV File (NFS 64521)

The **JadeProfiler** class now provides the Boolean **reportInCSVFormat** property that can be used to specify that the report is output to a list of the called methods with the execution times as comma-separated values. By default, the value is **false**; that is, the method call summary reports are directed to a log file. (You can also set the value of this property by using the new **ReportInCSVFormat** parameter in the [JadeProfiler] section of the JADE initialization file.)

You can now also request a report in CSV format by selecting the new **Report CSV** command from the Jade User Interrupt **Profiler** submenu.

Note The profile file handling has also changed. You can specify a file name in the **JadeProfiler** class **fileName** property or in the **ResultsFile** parameter in the [JadeProfiler] section of the JADE initialization file. In earlier releases, if a file name was not specified, the report file was called **JadeProf.log**.

The default file name is now dynamic, in the format **JadeProfiler_<*application-name*>_*yyyyMMdd_hhmmss* .log** or **.csv**. A new report file is therefore generated for each call to the **JadeProfiler** class **report** method.

Tip If the **ResultsFile** parameter in the [JadeProfiler] section of your JADE initialization file currently has a specified value, set the value to **<default>** or delete the parameter, so the next time you profile methods, the file parameter has the new name (otherwise the existing value is used). In addition, set the **MethodCount** parameter to **<default>** or delete the parameter, to ensure that the new method count is used.

JadeReportWriterReport Class Method Signatures (PAR 65262)

In earlier releases, the documentation for the **JadeReportWriterReport** class stated that the value of the **detailTag** parameter in the **getXmlOptions** and **setXmlOptions** methods was **Class** type.

This was incorrect, as the value of the **detailTag** parameter is **String** type.

JadeTestCase Class Assert Methods (PAR 64825)

The type of the parameters for the expected and actual values of the **JadeTestCase** class **assertEquals** method are declared as **Any**, which provides a general API that can be used to test for values of all JADE data types. However, this means that the compiler cannot do any type checking of the values. In JADE, two type **Any** variables are always different if the type of the value differs; for example,. **a1** is not equal to **a2** if **a1** and **a2** have different types.

In the example **assertEquals(1, 1.Decimal)**, the first parameter (**1**) has type **Integer** with a value of **1** while the second parameter (**1.Decimal**) has type **Decimal** with a value of **1**, so the assert will always fail. The assert should be coded as follows.

```
assertEquals(1.Decimal, 1.Decimal);
```

The message output by the **JadeTestCase** class **assertEquals** method if the assert fails because of the operands having different types has now been changed to make this clear; that is:

```
assertEquals - types differ, expected type-of-m value value-of-m but actual
type-of-n value = value-of-n
```

The **JadeTestCase** class **assertEqualsMsg** method has also been changed accordingly.

JadeTestRunner Class Methods

The **JadeTestRunner** class now provides the following methods.

- `setDebugOnAssert(value: Boolean) updating;`

  The **setDebugOnAssert** method specifies that the test runner invokes the **Process** class **debug** method if an assert fails.

- `setDebugOnException(value: Boolean) updating;`

  The **setDebugOnException** method specifies that the test runner invokes the **Process** class **debug** method if an exception occurs.

- `setDebugOnUnexpectedException(value: Boolean) updating;`

  The **setDebugOnUnexpectedException** method specifies that the test runner invokes the **Process** class **debug** method if an unexpected exception occurs. This method enables you to debug exceptions other than any exception that the test has registered with the **JadeTestCase** class **expectedException** method.

- `setLogCallStack(value: Boolean) updating;`

  The **setLogCallStack** method specifies that the test runner reports the call stack when a test method assertion fails or an exception is raised.

Load Factor Default Value of Sets (PAR 64939)

In earlier releases, the default sequential load factor of sets resulted in the creation of an unbalanced tree with low occupancy when data was added to a set.

The default value for the **loadFactor** property for **Set** subclasses is now random.

Package Initialization (NFS 65116)

In earlier releases, packages were always initialized for all applications run from any schema.

JADE no longer initializes any packages for **RootSchema** applications running in user schemas, including unit tests run using the JADE Unit Test framework and the default RPS **Datapump** application in the **RootSchema**.

If package initialization is required, this must be done explicitly in your user code (using the **Process** class **initializePackages** and **finalizePackages** methods).

Presentation Client Installer has been Deprecated

As notified in JADE 7.1.09, the Customizable Presentation Client Installer (CPCI) has been deprecated.

Process Class userExitCode Property (NFS 43186)

In earlier releases, you could not set a user exit code for **jade.exe** when running over a thin client connection.

The **Process** class now provides the **userExitCode** property, which has an **Integer** value. This property contains a value returned by your applications when the **jade.exe** program exits. The default value is zero (**0**).

You can use this property, for example, to set a non-zero exit code that can then be checked in a batch file by using the **ERRORLEVEL** keyword to check for appropriate **userExitCode** values, as shown in the following example.

```
begin
    beginTransaction;
    process.userExitCode := 123;
    commitTransaction;
    terminate;
end;
```

The specified value is returned only if the JADE program would have normally returned zero (**0**); that is, if JADE wants to return a non-zero exit value, the JADE value takes precedence over your value specified in this attribute.

Note As the **userExitCode** property applies to the **jade exe** node, any JADE application executing from that same client environment can set this value. Cooperation between applications wanting to set this property may therefore be required.

This property gets or sets the exit code for the running **jade ex**e client. The property can be accessed only on the running process. Accessing it on a non-**jade exe** client or a process other than the current process of the application results in exception 1265 *(Environmental object operation is out of scope for process)*.

If the client is running as a standard client, the method is equivalent to **node.userExitCode** for the current processes node.

If the client is running as a presentation client, the exit code is retrieved or set on the **jade.exe** program of the presentation client.

Note As the process object is persistent, you must be in transaction state to set the value.

Relational Population Service (RPS)

This section describes the Relational Population Service (RPS) changes in the JADE 2016.0.02 release.

RPS Datapump Mapping Methods (PAR 65571)

In releases prior to JADE 7.1.10, the RPS **Datapump** application did not handle exceptions raised by Binary Large Object (blob), String Large Object (slob), and StringUtf8 Large Object (slobutf8) mapping methods.

From the JADE 7.1.10 and 2016.0.02 releases, the JADE **Datapump** application handles blob, slob, and slobutf8 mapping method exceptions.

Any exceptions produced are handled if the **NullColumnOnException** parameter in the [JadeRps] section of the JADE initialization file is set to **selected** so that the methods specified in the **Method<*n*>** parameters in the [RpsIgnoreMethodExceptions] section of the JADE initialization file allow unhandled exceptions to occur without the **Datapump** application terminating. (The column of the RDBMS corresponding to the method receives a null value, in this case.)

RPS Keywords (PAR 64608)

In releases prior to JADE 7.1.08.014, reserved T-SQL keywords (for example, the word **'file'** in the query **UPDATE BKdocument SET file=? WHERE oid = ?**) were not quoted in RPS.

The third sheet of the System DSN wizard in the ODBC Data Source Administrator (the Data Sources (ODBC) program in the Windows Administrative Tools) now has the configurable **Use ANSI quoted identifiers** check box, which enables RPS to interrogate the RPS ODBC data source that it uses to determine if to quote SQL identifiers (that is, table, column, and procedure names). This allows reserved T-SQL keywords to be used in SQL statements, which is useful if you want a property to have the same name as a T-SQL reserved word.

RPS OID Mapping (PAR 64633)

The JADE 2016.0.01 "Relational Population Service OID Mapping (NFS 62231)" topic in the JADE 2016.0.01 Release Information stated "As the maximum length of JADE entities increased from 30 to 100 in JADE 7.1, JADE now supports two RPS object identifier (oid) length mapping options: Map to String (7.0 format) and Map to String (7.1 format). The 7.0 format size is 16, and the 7.1 format size is 28."

This is incorrect, as JADE supports only 7.1-style mappings from JADE 2016.0.01 and later.

Renaming Global and Class Constants (NFS 64885)

The optional **commandFile** parameter in the batch JADE Schema Load utility (**jadloadb**) now enables you to rename global constants and class constants.

The following commands (each one on a separate line in the JADE Command File, or JCF) are:

```
Rename Constant schema-name::class-name::existing-constant-name new-constant-name

Rename GlobalConstant schema-name::existing-global-constant-name
new-global-constant-name
```

The source of any methods and constants that reference a renamed constant are updated to include the new constant name.

To programmatically rename one or more global constants, create a command file with the relevant commands and then use the JADE Schema Load utility to process your file. (For details about the syntax and command examples, see "commandFile" under "Loading a Schema and Forms in Batch Mode", in Chapter 1 of the *JADE Schema Load Utility User's Guide*.)

Reorganization Directories (PAR 65531)

In earlier releases, the reorganization work and backup directories for a file with a specified path did not default to the location of the file if it had been set using the batch JADE Database utility **setFilePath** command. The locations specified in the **ReorgWorkDirectory** and **ReorgBackupDirectory** parameters of the JADE initialization file defaulted to the specified database directory, which meant that when a file or a file partition had a specified path (for example, when locating it on another volume), a reorganization operation copied files across volumes (.**dat** files to .**bak** files and .**reo** files to .**dat** files).

You can now specify the token **<filepath>** in the JADE initialization file for these reorganization directories; for example:

```
[JadeReorg]
ReorgWorkDirectory=<filepath>
ReorgBackupDirectory=<filepath>
```

When you specify the **<filepath>** token for the reorganization:

- Work directory, the .**reo** files are located with the database (.**dat**) files
- Backup directory, the .**bak** files are located with the database (.**dat**) files

Report Writer Documentation Format (PAR 64676)

In earlier releases, the *JADE Report Writer User's Guide* was displayed in print (PDF) format only; for example, when selecting the Help menu **User's Guide** command of the JADE Report Writer Configuration or Designer application.

The **UseJadeWebHelp** parameter in the [JadeHelp] section of the JADE initialization file now also applies to help requested while running a JADE Report Writer application. If the value of the **UseJadeWebHelp** parameter is **true** (the default value), Report Writer help now displays the web (HTML5) format rather than the **ReportWriter.pdf** file.

See also "Using the JADE Report Writer White Paper", later in this document.

REST Services

This section describes the REST Service changes in the JADE 2016.0.02 release.

Generated REST Service Description (PAR 64969)

In earlier releases, the description generated from a REST Service class wrongly included all of the subclasses of the classes referenced in the **JadeRestService** subclasses communication methods. This produced a file that included a very large number of other classes.

The description now includes only the classes and the superclasses referenced by the specified parameter values and return types of the communications methods of the **JadeRestService** subclass selected for the application. Note, however, that no class description is included if those methods return only **Any** or a primitive type.

REST Services Exception Handling (PAR 64994)

In earlier releases, a REST service returned error 500 instead of error 400.

The exception handling for REST services has been changed so that it now identifies which part of the service is responsible for the issue, and returns to the client an HTTP error 400 (bad request from the user) or 500 (server error).

A 400 error is returned, for example, if the:

- URL is empty
- JSON or XML syntax is invalid

- Called method is not found
- Called method is protected or it is a type method
- Data for a method parameter is invalid for its type
- Data does not match the method signature
- Type of the object passed does not match the method parameter object type

A server error is returned for other failures such as a logic exception.

Running Unit Tests

This section describes the JADE Unit Test changes in the JADE 2016.0.02 release. (See also "JadeTestRunner Class Methods", earlier in this document.)

Pausing when an Unexpected Exception is Encountered (NFS 65344)

The File menu in the Unit Test Runner form now provides the **Debug on Unexpected Exception** command, which determines whether the unit test is paused and the call stack is displayed when the unit test encounters an unexpected exception. This command enables you to debug exceptions other than any that the test has registered with the **JadeTestCase** class **expectedException** method.

By default, this command is not checked.

This **DebugOnUnexpectedException** option and the **DebugOnException**, **DebugOnAssert**, **CodeCoverage**, and **IncludePassedTests** options are saved in the [JadeUnitTestRunnerUI] section of the JADE initialization file.

To toggle the display of the call stack for failed messages, select the **Debug on Unexpected Exception** command again, to remove the check mark and run all of the unit tests without pausing.

You can also double-click on a method that failed with an exception in the **Results** pane, to display call stack information.

The behavior of the existing **Debug on Exception** command in the File menu is unchanged; that is, it pauses if *any* exception occurs.

Running a Test Case with a Set Skin (PAR 65263)

In JADE 2016.0.01, when running a Web service consumer test case with some skins (for example, Cashmere) display issues resulted in menu item text being is unreadable because of the contrast between the menu background and foreground colors.

When a form skin has no menu skin attached to the form's skin, the menu colors are taken from the form's skin menu line properties. With some of the skins, these values have been set to handle the menu line having a dark background image. These values are not suitable for the menu windows themselves, as the text color is too close to the background color. This does not happen when the application is not the JADE development environment, because of specific handling.

For details about the change to skin handling, see "Skins (PAR 65263)", later in this document.

Running Unit Tests as a Fat or a Thin Client in Batch Mode (PAR 65202, NFS 64920)

In earlier releases, the "Running Unit Tests in Batch Mode" topic in Chapter 17 of the *JADE Developer's Reference* did not specify that the default value of the optional **runAllTests** attribute **<schema>** XML element is **false**.

For details about using additional command line parameters when running the applications as a standard client or thin client using the JADE executable (jade.exe), see "Additional Command Line Handling for Applications Run using jade.exe (NFS 64920)", earlier in this document.

You can now run the **RootSchema** non-GUI client **JadeUnitTestBatch** application as a standard (fat) or a presentation (thin) client using **jade.exe**. The application parameters specified on the command line must be delimited by the **StartAppParameters** and **EndAppParameters** arguments.

The following example specifies the JADE executable (**jade.exe**) command line for the non-GUI **JadeUnitTestBatch** application.

```
jade path=c:\jade\system schema=TestUnitTestSchema ini=c:\jade\system\jade.ini
app=JadeUnitTestBatch AppServer=MyAppServer AppServerPort=1234 StartAppParameters
fileName="g:\temp\manifest today.txt" report codecoverage=true output=true
g:\temp\UnitTest.log EndAppParameters
```

Running Unit Tests that Require GUI Components in Batch Mode (NFS 65222)

In earlier releases, you could not run unit tests that required GUI components in batch mode.

The RootSchema now provides the **JadeUnitTestGuiNoForms** application that you can use to run unit tests that require GUI components (that is, it does not display forms) in batch mode. You can run this application using **jade.exe** with the same parameters as the **JadeUnitTestBatch** application. These parameters are delimited by the **StartAppParameters** and **EndAppParameters** arguments.

For details about using additional command line parameters when running an application as a standard client or thin client using the JADE executable (**jade.exe**), see "Additional Command Line Handling for Applications Run using jade.exe (NFS 64920)", earlier in this document.

The following example specifies the JADE executable (**jade.exe**) command line for the **GUI No Forms JadeUnitTestGuiNoForms** application.

```
jade path=c:\jade\system schema=TestUnitTestSchema ini=c:\jade\system\jade.ini
app=JadeUnitTestGuiNoForms AppServer=MyAppServer AppServerPort=1234
StartAppParameters fileName="g:\temp\manifest today.txt" report codecoverage=true
output=true g:\temp\UnitTest.log EndAppParameters
```

Unit Test Batch Options (NFS 65075, PAR 65094)

In earlier releases, you could not run batch unit tests of methods that were checked out to a delta. In addition, the batch unit test runner did not display the call stack if a test failed due to an unexpected (unhandled) exception.

When you use the **jadclient** non-GUI client application to automate the batch running of unit tests, you can now specify the optional **<options>** XML element. You can use this to specify multiple options; for example, **<options deltaName="delta name" logCallStack="true" />**.

The **deltaName** option indicates that unit tests and methods checked out to the specified delta should be run.

The **logCallStack** option indicates whether the call stack is logged when an exception occurs. By default, the call stack is *not* logged; that is, **logCallStack="false"**.

The control file now has the following format.

```
<unitTest>
    <reporter schema="schema name" class="class name" logFile="file name"/>
    <schema name="test schema name" />
    <options deltaName="delta name" logCallStack="true"/>
    <tests>
        <test name="test name">
        ...
        </test>
    </tests>
</unitTest>
```

The following example shows the structure of a typical **<test>** XML element.

```
<unitTest>
    <options deltaName="dawn" logCallStack="true"/>
    <tests>
        <test name="MyTests">
        <schema name="TestUnitTestSchema"/>
        <class name="UnitTests" includeSubclasses="true"/>
        <class name="WithdrawalTests>
            <method name="overdrawn"/>
            <method name="exceedsLimit"/>
        </class>
        </test>
    </tests>
</unitTest>
```

Unit Test Method Options (NFS 65257)

You can now define a method that is called before any tests are run (to initialize any global data) and a method that is called after the last test has been run (to clean up), by using the respective new **unitTestBeforeAll** and **unitTestAfterAll** method options in a unit test method.

unitTestBeforeAll Method Option

The **unitTestBeforeAll** method option indicates that the method is a unit test method that is run once before any classes are tested.

Note Only one method in the classes being tested can have the **unitTestBeforeAll** option.

If a **unitTestBeforeAll** method fails, the first class method executed is counted as failed.

A method with the **unitTestBeforeAll** option specified can perform any one-off global initialization for all tests; for example, package initialization and creating test data.

unitTestAfterAll Method Option

The **unitTestAfterAll** method option indicates that the method is a unit test method that is run once after all classes have been tested.

Note Only one method in the classes being tested can have the **unitTestAfterAll** method option.

If a **unitTestAfterAll** method fails, the last class method executed is counted as failed.

A method with the **unitTestAfterAll** option specified can perform any one-off global clean up for all tests; for example, package finalization and deleting test data.

Unit Test Runner Application (PAR 64797)

If the unit test runner is already running from a schema in earlier releases, a new unit test runner application was started when unit tests were run from another schema.

To enable you to distinguish the application in which a JADE unit test is running, the Unit Test Runner form caption now displays the database path and the schema.

Skins

This section describes the skins and **JadeSkinControl** subclass changes in this release. (See also "Alternating Colors for List Box and Table Rows (NFS 65216)", earlier in this document.)

List Box and Table Skins (PAR 65752)

Additional skins features have been added to the **JadeSkinListBox** and **JadeSkinTable** classes, as follows.

- The back ground and text color of selected list box and table items,
- The skin to display when a **Table** cell has an **inputType** of **Table::InputType_CheckBox** or a **CheckBox** control is assigned as a cell control.

Note These features have been added as dynamic properties so that no reorganization is required to your existing features.

The following properties are defined in JADE 2018.0.01 as normal properties (no dynamic in the name) and the values for the dynamic properties will be transferred to the applicable 2018.0.01 properties when you upgrade to that release. The dynamic properties will not subsequently be used and will be removed in a future release.

In JADE 2016 releases, the **JadeSkinListBox** and **JadeSkinTable** classes now provide the:

- **selectionColorTextDynamic** property, which controls the text color of a selected item in a skinned list box or table.

  If this value is the default value of black (**0**) or it is **#80000000** (that is, transparent), the default selection text color defined by Windows is used.

- **selectionColorDynamic** property, which controls the color that is used to draw the background color of a selected item in a skinned list box or table.

  If this value is the default value of black (**0**) or it is **#80000000** (that is, transparent), the default selection text color defined by Windows is used.

  The default value of **#80000000** means that the default selection background color defined by Windows is used.

In addition, the **JadeSkinTable** class now provides the **myCheckBoxSkinDynamic** property (type **JadeSkinCheckBox**), which specifies the check box skin that is used when drawing a cell that has the **Table** class **inputType** property set to **InputType_CheckBox** or a cell control set to a **CheckBox** control. The value for a check box skin is defined in the **Table** control types on the **Controls** sheet of the Jade Skin Maintenance dialog. The default value of **<none>** indicates that check boxes in table cells are drawn without a skin.

Set the values for the **ListBox** and **Table** control types on the **Controls** sheet of the Jade Skin Maintenance dialog. The default selection text and background colors are **true**, by default; that is, the **Default Selection Text Color** and **Default Selection Back Color** check boxes are checked.

When the selection text or background color is not set to the default value or you uncheck the **Default Selection Text Color** or **Default Selection Back Color** check box, click the **Set** button at the right of the check box. The common Color dialog is then displayed, to enable you to select or define a custom text or background color for selected list box or table items.

Menu Skins (PAR 65263)

Skin handling has now been changed so that if a menu skin is defined with no values set (defaults) and assigned to the form skin, the menus are drawn without using skin elements. This mechanism provides a way of having menus drawn in the default non-skinned manner but still allows the menu line to be skinned.

The default settings are:

- No border, separator, right arrow, and check mark image colors all set to **default**
- Line height and space before the check mark are zero (**0**)
- Space after the check mark, after the picture, and before the accelerator are **5**

In other words, if the menu skin associated with a form's skin has only default values set, the menu is not drawn using the menu line properties on the form skin but is drawn in the default non-skinned manner.

See also "Running a Test Case with a Set Skin (PAR 65263)" under "Running Unit Tests", earlier in this document.

Searching for a String in a Control

The **ComboBox** and **ListBox** control classes now provide the following methods, which search the entries in a combo box or list box for a case-sensitive entry that matches the string specified in the **str** parameter or a case-sensitive entry in which the string exactly matches the string specified in the **str** parameter, respectively.

```
findStringCaseSensitive(startIndx: Integer; str: String): Integer;

findStringExactCaseSensitive(startIndx: Integer; str: String): Integer;
```

The **JadeTableColumn** class now provides the **findString** method, which has the following signature.

```
findString(str: String; row: Integer io; caseSensitive: Boolean; exact: Boolean):
Boolean;
```

This method searches the cells in a column of a **Table** control for the string specified in the **str** parameter. The search starts at the cell row specified by the **row** parameter or at the first row if the parameter value is less than **1**.

The **JadeTableRow** class now provides the **findString** method, which has the following signature.

```
findString(str: String; column: Integer io; caseSensitive: Boolean; exact:
Boolean): Boolean;
```

This method searches the cells in a row of a **Table** control for the string specified in the **str** parameter. The search starts at the cell column specified by the **column** parameter or at the first column if the parameter value is less than **1**.

The **JadeTableSheet** class now provides the **findString** method, which has the following signature.

```
findString(str: String; row: Integer io; column: Integer io; caseSensitive:
Boolean; exact: Boolean): Boolean;
```

This method searches the cells in a sheet of a **Table** control for the string specified in the **str** parameter. The search starts at the cell row and column specified by the respective **row** and **column** parameter values or at the first row and column if the parameter values are less than **1**.

String Replacement Functionality (NFS 65528)

JADE now provides two new external methods: **stringReplace** and **stringReplaceFrom**.

Both methods return a new string that has had the replacement made. The receiver string has the substring specified by the **target** parameter replaced by the substring specified in the **replacement** parameter. The **replace** method replaces all occurrences; the **replaceFrom** method replaces only the first occurrence starting from the specified **startIndex** parameter.

As these are provided in a service pack, they have not yet been defined in the **String** primitive type, to prevent name clashes. If you want to make use of them, implement the following external methods, bearing in mind you can call the JADE methods whatever you like. These methods will be implemented in the **String** primitive type in the JADE 2018 release, with the following names and signatures. You will need to rename any methods defined in the **String** primitive type that will clash with the following methods before you upgrade to JADE 2018.

```
replace(target: String;
        replacement: String;
        bIgnoreCase: Boolean): String;

replaceFrom(target: String;
            replacement: String;
            startIndex: Integer;
            bIgnoreCase: Boolean): String;
```

These methods are called **stringReplace** and **stringReplaceFrom** in the **jomsupp** library.

Synchronized Database Service (SDS)

This section describes the Synchronized Database Service (SDS) changes in the JADE 2016.0.02 release. (See also "Archival Recovery (PAR 61735)", earlier in this document.)

SDS Replay Record Sequence Exception (PAR 65483)

In earlier releases, a record sequence exception caused an SDS replay failure.

When a primary changes to archive mode, the SDS service, if active, is now stopped. The service is restarted as necessary when exiting from archive mode.

SDS Database Role (PAR 63603)

In earlier releases, the JADE Database utility batch **clearRole** command was performed on a backup of a secondary system leaving it in an inconsistent state.

It is generally unsafe to clear the role of a secondary system, as they almost always require recovery. You should clear the role by specifying the **clearSDSRole** parameter in the recovery from backup. If you do *not* specify the **clearSDSRole** parameter in the recovery from backup, the recovery operation is recovering a secondary database as a secondary. This means that:

1. No undo will be performed
2. A partial final journal will not be processed (when the secondary initializes, it re-fetches the journal from the primary)

The expectation is that the database will connect to the primary and then synchronize. When the last journal is not applied when recovering from a backup, the backup recovery is incomplete and file integrity is not restored. The undo action is suppressed, so there are further complications if there are incomplete transactions.

This has been corrected. The **clearRole** command is no longer permitted on SDS Secondary systems, and the **jdbutil** and **jdbutilb** executables **showInfo** command now always shows SDS Secondary systems as requiring recovery.

Secondary Database Reconnect Interval (PAR 65510)

The **ReconnectInterval** parameter in the [SyncDbService] section of the JADE initialization file now states the following.

The **ReconnectInterval** parameter specifies the frequency at which a secondary database server attempts to reconnect to its primary server when a primary server is not available.

When the connection to the primary fails, a reconnection is attempted after 10 seconds. If this fails, reconnection attempts are performed using the number of seconds specified in this parameter. (The default value is 300 seconds; that is, 5 minutes.)

The parameter is read when the secondary to primary connection is next established; for example, you can disable the connection, change the parameter value, and then reconnect.

Shutting Down a Secondary with Incomplete Transactions (PAR 65106)

Error 1284 *(User sign-on is currently disabled)* is raised when an application attempts to sign on while signing on is disabled. If this occurs when you are attempting a hostile takeover (for example, if a secondary database is shut down with incomplete transactions and has not been able to establish communications with its primary), you can now sign on to the SDS Administration application in single user mode and then perform a hostile takeover.

If the primary was unavailable because of a network outage that has been resolved, try to sign on again later, after signing on has been re-enabled.

Timing Out Response from a Secondary (NFS 65076)

In earlier releases, you could not configure the maximum time that a primary waits for a response from a secondary.

The [SyncDbService] section of the JADE initialization file can now contain the **ResponseTimeout** parameter, which is configured on primary databases. This parameter defines the maximum length of time the primary waits for a request response from a secondary.

The parameter is read when the SDS primary service is next initialized. You can start and stop the SDS service without restarting the node.

The default value is 60 seconds, the minimum value is 10 seconds, and the maximum value is 120 seconds.

If the timeout expires, the request is terminated with error 3212 (*SDS a response was not received within a reasonable timeframe*). If the secondary disconnects while the primary is waiting for a request response, the request is terminated with error 3204 (*SDS secondary not attached*).

White Paper Additions

The JADE 2016 product information library now contains the following white papers.

- REST Services
- Using the JADE Report Writer

REST Services White Paper

The JADE 2016 product information library now contains the *REST Services White Paper*, which contains information about the REST-based Web services that JADE provides.

Using the JADE Report Writer White Paper

The JADE 2016 product information library now contains the *Using the JADE Report Writer White Paper*, which provides an understanding of how to use the JADE Report Writer to improve performance during execution of user reports.

Changes and New Features in JADE Release 2016.0.01

This section summarizes the product and documentation changes and new features in JADE release 2016.0.01. For details about the changes in release 2016 that may affect your existing systems, see "JADE 2016 Changes that May Affect Your Existing Systems", earlier in this document. See also "Binary Type pos Method Deprecation" under "Deimplementations and Deprecations".

.NET

This section describes the .NET changes in this release.

.NET Decimal Conversion (PAR 63449)

When converting a Common Language Runtime (CLR) Decimal data type to a JADE **Decimal** primitive type in earlier releases, the wrong value was written to a JADE property from a .NET or ActiveX decimal if the value contained 29 significant digits. The algorithm used to round a decimal value was wrong if the value of the number represented in the significant digits, ignoring the decimal point, was greater than or equal to 2^95 (39,614,081,257,132,168,796,771,975,168), which affected .NET and ActiveX decimal values.

Appendix A of the *JADE .NET Developer's Reference* stated that when converting a CLR Decimal data type to a JADE **Decimal** primitive type, there may be an overflow or data loss saving data to the database. From JADE 7.0.11 and 7.1.07, this was no longer true. If necessary, JADE rounds decimal values on assignment to the number of decimal places defined on the property.

.NET Import and Usage (PAR 63621)

In earlier releases, scaling issues could occur when a .NET control was loaded, as Windows assumed that JADE was not dots per inch (dpi)-aware. JADE is dpi-aware if the value of the **Form** class **scaleForm** property is **true**, so that JADE scales the forms according to the current system dpi. However, when a .NET assembly is subsequently loaded, the Microsoft dpi-awareness handling is invoked and it is assumes that the forms need to be rescaled using its virtualization philosophy, which distorts the displayed forms.

On Windows 8.1 and above, JADE now informs the operating system that it is dpi-aware (system-wide; not the scaling of each dpi monitor), which prevents Windows from attempting to scale again.

Accessing a .NET Exception Object (NFS 64046)

The **JadeDotNetInvokeException** user interface exception class now provides the **dotNetExceptionObject**, which is a type of **JadeDotNetType**. This property is populated with the .NET **Exception** object when an exception of type **JadeDotNetInvokeException** is generated and the class of the .NET exception object class was imported from the .NET assembly.

If the exception type object is not available, the property value is null. If the class was not imported from the .NET assembly, an object of **JadeDotNetType** type is created.

If the property is set to a reference, you can cast the object to its type and use it to obtain further .NET exception information.

Bulk Deselection of .NET Classes to Import (NFS 63569)

The .NET Import Wizard sheet that enables you to specify the names for each class that is created now provides a button that enables you to toggle whether all classes in the imported .NET assembly are excluded or included. (In earlier releases, you had to select each class to be excluded individually and then check the **Don't import this** check box.)

This button enables you to mark all classes as being excluded and then you can individually check only those classes that you want to import.

When you click the **Mark all classes as being excluded** button, the caption changes to **Mark all classes as being imported**. Clicking the button a second time marks all classes as being included, again.

Note As you cannot import a class unless its superclasses are also imported, after you click the button to exclude all classes, you must select each superclass of the classes to be imported and then uncheck the **Exclude** check box in the table at the right of the next sheet of the wizard.

Static Method Handling (NFS 60015)

In earlier releases, JADE could not call a factory method on a C# class whose constructor is not public.

You can now import static methods on .NET classes into JADE as type methods. (Type methods can be called without having an instance of the class, which enables you to call factory methods directly without the need to wrap the class in a wrapper class.)

For details, see "Type Methods (NFS 63225)" under "JADE Development Environment", later in this document.

Accessing a Merged View of Set Instances (PAR 61277)

JADE now supports sequential access of objects from a merged view of two or more **Set** instances, to enable you to iterate over a number of sets. The **Set** instances do not need to have the same membership. (In earlier releases, you could sequentially access objects from a merged view only of compatible **Dictionary** instances.)

The new **SetMergeIterator** class (a subclass of the **Iterator** class) implements behavior equivalent to that of the **MergeIterator** class over dictionaries.

The methods defined in the **SetMergeIterator** class are summarized in the following table.

Method	Description
addCollection	Adds the specified set to the merged iterator view
back	Accesses entries in reverse order in the merged iterator view
current	Returns the last value iterated by the **back** or the **next** method
getCollectionAt	Returns the set at the specified index in the collection of sets making up the merged iterator view
getCollectionCount	Returns the number of sets
getCurrentCollection	Returns the set containing the last value iterated by the **back** or the **next** method
isValid	Returns **true** if the receiver is a valid iterator
next	Accesses successive entries in the merged iterator view
removeCollection	Removes the specified set from the merged iterator view
reset	Initializes the iterator
startAtObject	Sets the starting position of the iterator at the position of the specified object

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

Binary::uuidAsString Method (PAR 62807)

When calling the **Binary** primitive type **uuidAsString** method, to be a valid Universally Unique Identifier (UUID), the binary should be 16 bytes. If it is less than 16 bytes, the value will be internally padded with zero bytes on the end, to make it 16 bytes long before the conversion is performed.

If it is longer than 16 bytes, exception 1091 *(Binary too long)* is raised.

CMDPrint::warnIfNoDefault Property (PAR 62950)

In earlier releases, the default value of the **CMDPrint** class **warnIfNoDefault** property was documented as **false**. However, the default value is **true**; that is, a warning message box is displayed by default if there is no printer default for the system on the common Print dialog.

Control Classes

This section describes the **Control** class and subclass changes in this release.

Adding New Properties to a Control (NFS 30651)

When new properties were added to a JADE control in earlier releases, they were not always added to the list of available properties that can be set in the JADE Painter for subclasses of these controls. If they were not added, you had to add them by using the Design Time Properties dialog, accessed by clicking the **Design Time Properties** button on the **Options** sheet of the Define Class dialog.

This process is now automated so that when upgrading to JADE 2016, any properties defined in the **Control** subclasses are added to any user subclasses of each control. In addition, when a forms definition (.**ddb**) load is performed for a user subclass, the list of development properties is also upgraded to include any properties not defined for the user subclass.

Note If you manually remove RootSchema properties from the list of properties available in the JADE Painter for a subclassed control, the upgrade process adds them back again.

Animating the Hiding or Showing of a Control (NFS 64250)

The **Control** class now provides the **animateWindow** method, which enables special animation effects (roll, slide, collapse, or expand) when showing or hiding a control.

Calling the method is equivalent to toggling the **visible** property of a control where the resulting showing or hiding of the control is animated. When the control is not visible, calling the **animateWindow** method makes it visible. If the control is visible, calling the **animateWindow** method makes it invisible.

For details and examples, see the *JADE Encyclopaedia of Classes*, Volume 3.

Auto-sizing CheckBox and OptionButton Controls (NFS 64290)

In earlier releases, the height of check boxes and option buttons was always set to the minimum required to show the content.

The **autoSize** property has now been implemented on the **CheckBox** and **OptionButton** controls. This property is **false**, by default.

When this is property is **true**, the **CheckBox** and **OptionButton** controls autoSize to fit the content; that is, the width is the size of the icon displayed plus the width of the text, and the height is the minimum required to show the content.

When this property is **false**, the control height and width are set in the JADE Painter or by logic, except if the height is less than size required to fit the content, in which case the height becomes the minimum required to show the content vertically.

In the new implementation, if the control height is larger than required, the content is centered vertically.

Docking Controls (NFS 32962)

The following new **Integer**-value properties defined in the **JadeDockBase** control class apply to the **JadeDockBar** and **JadeDockContainer** control subclasses.

- **maximumHeight**
- **maximumWidth**
- **minimumHeight**
- **minimumWidth**

You can set the value of these properties in development and at runtime, to specify the maximum and minimum number of pixels for the height and width of dock controls. For details, see the *JADE Encyclopaedia of Classes*, Volume 3.

Focus Color of a Control (NFS 33462)

The following Integer-value properties have been defined in the **Control** class.

- **focusBackColor**
- **focusForeColor**

The properties are available in development (on specific controls only) and at run time.

The default value of zero (**Black**) indicates that the property is always ignored when drawing the control.

When the value of the **focusBackColor** property is not **Black**, that property value is used instead of the value of the **backColor** property to erase the control area when the control has focus or a child of the control has focus.

Note If the control is transparent, the value of the **focusBackColor** property is not used. (The control area is not erased as part of the painting of the control.)

When the value of the **focusForeColor** property is not **Black**, that property value is used instead of the value of the **foreColor** property to draw the text associated with the control when the control has focus or a child of the control has focus.

You can use the **focusBackColor** and **focusForeColor** properties to give the user a better visual prompt as to which control has focus.

Although the properties are defined in the **Control** class, they are not relevant to all controls. The controls that make use of these properties must be capable of gaining the focus, they can be control parents, and they cannot be external controls such as .NET controls.

For details, see the *JADE Encyclopaedia of Classes*, Volume 3.

JadeRichText Control (NFS 62749)

You can now set the background color, the font underline type, and whether selected text is drawn as a URL in a **JadeRichText** control, by using the following **JadeRichText** control properties.

- **selBackColor**, which specifies the color of the background of the currently selected text.
- **selFontUnderlineType**, which specifies the underline style of the currently selected text (that is, one of none, line, dash, dash-dot, dash-dot-dot, dotted, thick, wave, and invert).
- **selLink**, which specifies whether the currently selected text is a link that will be drawn as a URL. If the user clicks this link, the existing **JadeRichText** control **linkClicked** event method is called, passing the text of the link.

For details (including code examples and new **Underline_Type_** constants defined in the **JadeRichText** class), see the **JadeRichText** control in the *JADE Encyclopaedia of Classes* (Volume 3).

Label Control Underline Suppression (NFS 63773)

You can now suppress the display of the prefix underline facility on a **Label** control, by specifying the new **Label** class **noPrefix** Boolean property, which enables you to control whether the character following a single ampersand (**&**) is underlined. The default value is **false** in development and at run time (that is, the behavior in earlier releases is retained).

When the value of the **noPrefix** property is **false**, the first character of a label that is proceeded by a single **&** character is underlined. Set the property to **true**, to display the **&** character but with no underline attribute applied to the following character.

mouseHover Event (NFS 63931)

JADE has now implemented the **mouseHover** event method for a number of control subclasses. For details, see the *JADE Encyclopaedia of Classes*, Volume 3.

The **mouseHover** event occurs when the user moves the mouse onto the control and then the mouse remains static for one second or longer.

Note This event can occur over the non-client parts of the control. The **x** and **y** positions are relative to the client area of the control and can be negative or greater than the client width and height.

The event can also occur if a mouse button is down and the Shift or Ctrl key is down.

With the implementation of this event, there may be cases where an existing **mouseMove** event can be replaced by use of the **mouseHover** event, as the **mouseHover** event can achieve what is required with one event instead of several **mouseMove** events.

Picture Class rotation Property (PAR 63420)

The description of the **Picture** class **rotation** property is updated to provide more information, as follows.

- The picture is sized according to the size of the picture and the value of the **stretch** property setting (ignoring the value of the **rotation** property).
- The picture is then drawn to that size and rotated about the central point of the control. Any parts of the picture outside the control are clipped.

As a result, the best way to handle the **rotation** property is to:

1. Construct the picture in another **Picture** control to the required size without rotation, with the appropriate **stretch** property value and control size.
2. Use the **Window** class **createPictureAsType** method to obtain the binary picture value from the **Picture** control that you created in the previous step of this instruction.

   If you call the **createPictureAsType** method with a bit value less than the bit value of the original image, color distortion can occur.
3. Resize the destination **Picture** control so that it fits the rotated picture. (Set the destination **stretch** property value to **Stretch_None** and the **picture** property to the binary value returned from **createPictureAsType** method.)

In addition, creating a picture with the **pictureType** method set to **PictureType_Jpeg** results in picture image quality reduction, because the JPEG format is lossy; that is, picture quality is reduced to lessen the size of the resulting image. To retain existing quality and produce a smaller binary image, use a picture type such as **PictureType_Png**, which has a lossless compression of images for greater clarity.

Note Any stretching of an image can cause image distortion.

---

resetFirstChange Method (NFS 62851)

The **Control** class now implements the **resetFirstChange** method, which resets the **firstChange** event status of the control and all children of the control. As only the **TextBox**, **JadeRichText**, and **JadeEditMask** controls have a **firstChange** event, all other controls ignore the method other than to call the method on any children.

For details and examples, see the *JADE Encyclopaedia of Classes*, Volume 3.

Setting Font Properties

The **Control** class now provides the **setFontProperties** method, which sets the value of the **fontName**, **fontSize**, and **fontBold** properties of the control in one action; that is, instead of defining the example shown in the following code fragment.

```
listBox1.fontName := "Arial";
listBox1.fontSize := 9;
listBox1.fontBold := true;
```

Using the **setFontProperties** method can be more efficient than setting the properties individually, because the three properties are all set in the same action. When each property is set individually in JADE logic, a new font is created each time, and any impacts that the changed font has on the control size are applied; for example, auto-sizing, **parentAspect** positioning, or aligning controls. For details, see the *JADE Encyclopaedia of Classes*, Volume 3.

Table Class showPartialTextBubbleHelp Property (NFS 33216)

When the user moves the cursor over a table cell for which the text is not fully visible, a bubble help window displaying the full text can now be shown.

This feature is controlled by the **JadeTableSheet** class **showPartialTextBubbleHelp** property, which is a **Boolean** primitive type that defaults to **true** and that is available at run time only; for example:

```
table.accessSheet(1).showPartialTextBubbleHelp := false;
```

When the value of this property is **true**, the bubble help is displayed when the cursor is over a cell in which the text is not fully visible. For details, exceptions to this, and when bubble help is hidden, see the *JADE Encyclopaedia of Classes*, Volume 2.

Table Column Sorting (PAR 63893)

The handling of the **Table** class **sortType** property that converts from a string to the specified sort type is based on the current locale used by the application and the value of the **EnhancedLocaleSupport** parameter in the [JadeEnvironment] section of the JADE initialization file.

In earlier releases in a Windows 10 environment, a change of the default time formatting for New Zealand to use AM/PM format meant that the conversion of a **String** primitive type to a **Time** primitive type failed using some JADE methods if the **String** time included the **'AM/PM'** format (in particular, sorting a **Table** control by a time-based column failed to sort correctly). JADE now handles an **'AM/PM'** string format.

In addition, the **Table** class sorting for **Date** and **TimeStamp** primitive type-based columns has also changed. In earlier releases, the strings were converted to a **Timestamp** or **Date** primitive type using the current locale format. If the string was not in the appropriate format, the sorting failed to produce the required sequence. If the conversion of the date part fails based on the locale format, JADE now attempts to determine the format (*d/M/y*, *M/d/y*, or *y/M/d*) based on the type of data (that is, year length = 4 and whether the month is alpha).

Table Handling (NFS 63870)

The following behavioral changes have been made to the **Table** control class.

- Clicking on an enabled fixed row or column cell now draws the cell in a slightly darker background color. When the mouse is released, the original background color is restored.

  This gives the user a visual indication that the cell was clicked.

- Clicking on a disabled cell no longer generates a **click** event after the mouse is released. The table **mouseDown** and **mouseUp** events are still generated.

  The reason for this is that when the **click** event is received, it appears as though the user clicked the cell that has focus rather than the disabled cell.

- Resizing a row or column no longer generates a **click** event after the mouse is released. The table **mouseDown**, **mouseUp**, and **resizeRow** or **resizeColumn** events are still generated.

  The reason for this is that when the **click** event is received, the logic is unaware that the user wanted only to resize the row or column rather than, for example, re-sort the table.

- Moving a row or column no longer generates a **click** event after the mouse is released. The table **mouseDown**, **mouseUp**, and **queryRowMove** or **queryColumnMove** events are still generated.

  The reason for this is that when the **click** event is received, the logic is unaware that the user wanted only to move the row or column rather than, for example, re-sort the table.

TextBox Handling (NFS 63772)

In earlier releases, if a form had an enabled and visible default button defined, pressing the Enter key while in a text box clicked the default button and the text box did not receive the Enter key.

The **TextBox** class now provides the **wantReturn** Boolean property, which defaults to **false** during development and at run time. When this property is set to **false**, entering a carriage return while the text box has focus and the form has a default button causes the default button to get focus and be clicked. If the form does not have a default button, the carriage return is passed to the text box.

When the **wantReturn** property is set to **true** and the text box has focus, a carriage return is always passed to the text box and any default button is unaffected.

Note This property is ignored in Web-enabled forms.

Copying Images to and Pasting Images from the Clipboard (NFS 63630)

The **Application** class now provides the following methods, which enable you to copy an image to or paste an image from the Windows clipboard. These methods enable you to:

- Copy an image loaded into JADE and paste it into another application (for example, Word for Windows)
- Paste an image from the Windows clipboard into JADE as an image entity instead of having to browse for an image that you have opened in another application

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

copyImageToClipboard

The **copyImageToClipboard** method copies the binary image specified in the **image** parameter to the Windows clipboard. The image must be a .**bmp**, .**jpg**, .**jp2**, .**tiff**, .**gif**, .**png**, or enhanced meta file type image.

Except for the enhanced meta file type, JADE converts the file to a bitmap image and pastes it to the clipboard using the standard Windows clipboard style of **CF_DIB**. (There are no standard format types available for the other file types.)

An enhanced meta file type is added to the clipboard as a standard Microsoft Windows **CF_ENHMETAFILE** clipboard style.

If the image is a .**png** file, an additional clipboard entry is made under a non-standard clipboard format named **PNG**. This format is used and accepted by some other applications.

Notes For the Windows **CF_DIB** style, Microsoft Windows usually creates additional clipboard copies of the image under other styles.

The rules regarding retrieving transparent images from the clipboard are vague, and applications have different implementations. The result is that the transparency can be lost by some applications, but not by JADE. For that reason, JADE adds a **PNG** clipboard entry for .**png** files so that transparency can be always retained if that format is used.

---

copyImageFromClipboard

The **copyImageFromClipboard** method performs the following actions to copy an image from the Windows clipboard if it is available in the Windows clipboard.

1. Checks whether a non-standard clipboard format called **PNG** is available. If it is available and the content is a .**png** file, a copy of that image is returned.
2. Checks whether an image in the standard Microsoft Windows **CF_DIB** format is available. If so, that image is copied from the clipboard and returned as a bitmap-type (.**bmp**) image.
3. Checks whether an image in the standard Microsoft Windows **CF_ENHMETAFILE** (enhanced meta file) format is available. If so, that image is copied from the clipboard and returned.

If the clipboard is locked by another Windows process (which is not a normal occurrence) or an image in any of the supported formats is not available, this method returns a null binary.

Date Handling (PAR 62888)

In earlier releases, the Thai calendar was the only non-Gregorian calendar handled by JADE.

JADE now handles any calendar available within Microsoft Windows. Non-Gregorian calendars are handled in the locale-based date routines such as **parseForCurrentLocale**, **longFormat**, **shortFormat**, and so on, as well as the **setTextFromDate** and **getTextAsDate** methods in the **JadeEditMask** class.

In addition, the **Date** primitive type **isLeapYear** method now uses the currently set locale and calendar of the user to determine whether the date is a leap year. (In earlier releases, this method returned whether the date was a leap year in terms of the effective Gregorian date.)

Decimal Array Scale Factor not Enforced (NFS 64143)

As the scale factor in decimal arrays is currently not enforced, the **Scale Factor** text box on the **Membership** sheet of the Define Class dialog is disabled when adding or maintaining a **DecimalArray** class.

Deltas

This section describes the delta changes in this release.

Delta Browser Display (NFS 63232)

The Delta Browser now displays delta information in a table, with columns displaying:

1. Whether the delta is the currently set delta (arrow picture else empty)
2. Delta identifier
3. Description

4. Name of the user who created or last updated the delta information
5. The creation timestamp of the delta
6. The number of checked out methods for the delta, and is updated when a method is checked out, unchecked out or checked in

The displayed deltas are sorted in ascending order of the delta identifier, by default. Clicking on a column in the fixed row of the table toggles to sort order.

The new **Search for Delta Id containing:** text box enables you to specify case-insensitive text and then click the **Search** button, to locate the next delta identifier (starting from the delta after the currently selected delta) that contains the specified text.

Delta Browser Functionality (NFS 63355)

The JADE development environment now handles deltas as follows.

- When a delta is set, the delta text displayed on the right of the JADE development environment toolbar is now drawn in red. When you double-click on this delta text description, the Delta Browser is displayed.
- Double-click on the currently set delta entry in the table in the Delta Browser to unset the delta.

Delta Identifier and Description Maintenance (NFS 63267)

You can now change the identifier and description of an existing delta, by clicking on a delta in the Delta Browser and then selecting the **Change** command in the Delta menu. The Change Delta Definition dialog, containing the identifier and description of the currently selected delta entry, is then displayed. You can update these values to suit your requirements, and then click the **OK** button.

Note If patch versioning is used, changing the name of a delta also updates the name of the delta in any JADE Patch Version detail entries for that delta.

Dragging and Dropping Files and Folders (NFS 33755)

Files and folders can now be dropped onto a window, by calling the new **Window** class **setDragAndDropFiles** method, which establishes the window as an allowed drop target. When files and folders are dropped onto that window or one of its children, JADE calls the method specified in the **method** parameter, passing the list of files and directories dropped. It is up to the specified method to process the list of files or directories.

Note When the user drags the files or folders over the window, Microsoft changes the cursor to a plus (**+**) symbol in a box, indicating that the window will accept dropped file and folders.

The method passed to the **setDragAndDropFiles** method must be a method defined on the current form or a method defined on the class of the targeted control (or a superclass of the current form or targeted control).

For details and an example, see the *JADE Encyclopaedia of Classes*, Volume 3.

Note You cannot drag and drop files onto external .NET or ActiveX controls using this mechanism.

Dynamic Property Clusters

This section describes the dynamic property cluster changes in this release.

Deleting Dynamic Clusters (PAR 62396)

As a dynamic cluster assigned to a class cannot be deleted if the class has instances, the **Delete** button on the Dynamic Clusters dialog is disabled if the owning class has instances, because the instances may still have data associated with deleted properties that were owned by the cluster.

Extracting and Loading Dynamic Property Clusters (PAR 62434)

In earlier releases, if a dynamic property cluster was assigned to a class but it contained no properties, the cluster was not extracted in the schema file and was lost if the schema was loaded into another database.

Defined dynamic property clusters are now included in the attributes of a class's definition in the schema, and are loaded regardless of whether the cluster has defined properties, which enables you to share code via schema extract and load actions.

External Function Calls (PAR 63250)

JADE now publishes external functions declared in the RootSchema **jomos** external function library that call Windows library functions as defined in the Microsoft Developer Network (MSDN). These external functions are declared to avoid having to declare ANSI- and Unicode-specific definitions for some commonly called Windows library functions; for example, **josShellExecute** calls **ShellExecuteA** in an ANSI environment and **ShellExecuteW** in a Unicode environment.

The following table maps functions defined in the RootSchema **jomos** external function library to the corresponding MSDN function.

JADE External Function	Windows Function	Description
josCreateDirectory	CreateDirectory	Creates a directory that inherits information from other directories (security attributes default to null)
josDeleteDirectory	RemoveDirectory	Deletes an existing empty directory
josFileAccess	GetFileAttributes	Retrieves attributes for a specified file or directory (for details, see the paragraph that follows this table)
josFileCopy	CopyFile	Copies an existing file to a new file
josFileDelete	DeleteFile	Deletes an existing file
josGetKeyState	GetKeyState	Retrieves the status of the specified key; that is, whether the key is up, down, or toggled
josGetLastError	GetLastError	Retrieves the last error code value of the calling thread
josShellExecute	ShellExecute	Opens or prints the specified file; for example, to start another program under Microsoft Windows

In addition to these functions, the **josValidateDirectory** external function is declared, which validates that the specified name is a directory. For more details, see Appendix B of the *JADE External Interface Developer's Reference*.

The **Window** class now provides the **getWindowHandle** method, which returns the Windows handle as a **MemoryAddress**.

Note If a node can execute as a 32-bit or 64-bit node, you should use this method rather than the **Window** class **hwnd** and **getHwnd** methods, to ensure that the correct Windows handle is used.

The signature of the **josShellExecute** method changed in JADE 7.1.07. Prior to JADE 7.1.07, the Windows handle parameter was declared as an **Integer**, and it is now a **MemoryAddress**. A similar change has been made to the unpublished **_getCurrentThread**, **_getThreadCPUTimes**, **_shellExecute**, and **_shellExecuteUnicode** methods. All of these functions now use a **MemoryAddress** rather than an **Integer** for the Windows handle, to ensure that the correct value is used for the currently executing build type (that is, 32-bit or 64-bit). The upgrade process from a JADE 7.1 release has been changed to include a check for methods that reference these functions. Any such references will result in the method being marked for recompile during the upgrade validation phase. You must correct the references before restarting the validation process.

You must manually check and correct any references in systems that are already running on a JADE 7.1 release.

FileFolder Class

This section describes the **FileFolder** class changes in this release.

browseForFolder Method (NFS 63418)

The dialog that is displayed when you call the **FileFolder** class **browseForFolder** method now includes the **Folder** text box below the directory list box, so that the user can specify the name of the required folder, using a standard (common) Microsoft dialog.

Multiple Masks (NFS 48321)

In earlier releases, one mask value only (for example, ***.txt**) was handled by the **FileFolder** class **mask** property.

From this release, multiple mask values are supported. To specify multiple masks, separate them using the vertical bar (|) character; for example, **"*.txt | *.log | *.cat"**.

The **FileFolder** class **files** method now returns an array of all of the files that match any of the **mask** property values. The order of files within the array depends on the operating system, but the files are grouped according to the list of mask values, because a separate pass over the folder's files is required for each mask value.

Note Specifying **"*.*"** as a mask value results in all files and subfolders in the folder being returned.

JADE Audit Access (NFS 63677)

The **JadeAuditAccess** class now provides the following methods.

- **getNextRecordUTC**, which returns the next (relevant) record retrieved from the current journal file with additional Coordinated Universal Time (UTC) timestamp information.
- **registerFilterTimeRangeUTC**, which specifies the first and last timestamps in UTC for filtering accessible audit records.

In addition:

- The **JadeDatabaseAdmin** class now provides the **getArchiveJournalDirectory** method, which returns the name of the archive directory for transaction journals.
- The *Audit Access White Paper* has been updated.

JADE Database Utility (NFS 63361)

The JADE Database utility Select Files dialog for various database operations (for example, for certifying the database) has been enhanced as follows.

- The dialog has been enlarged, and it can now be resized.
- The dialog now provides the **Sort by** combo box that enables you to select the way in which database files

listed in the **Database Files** list box are sorted.

By default, the files are sorted by database file number, but you can sort them by database file name or by directory and then file name within each directory.

- The column widths of the text in the **Database Files** and the **Selected Files** list boxes have been enlarged to provide a better display for longer database file names.

In addition, the Open Database dialog and Select Directory dialog have been enlarged to better present longer directory names.

JADE Development Environment

This section describes the JADE development environment changes in this release. (See also "Bulk Deselection of .NET Classes to Import", earlier in this document.)

ActiveX Import Wizard (NFS 63096)

The ActiveX import wizard has been changed as follows.

- The form can now be resized and maximized.
- On the sheets that have a table, there is now a **Find in Table** text box and a **Find Next** button that enable you to search for text in the displayed table.

  Entering text in the **Find in Table** text box and then clicking the **Find Next** button searches the displayed table for the specified text. The search is case-insensitive.

  If the search string is found within a cell, the cell is then displayed with red text and the font increased in size to highlight the located entry. Click the **Find Next** button again to search for the next cell with that text.

  If the search text is not found in subsequent cells, a message box is displayed.

  Changing the text or pressing the **Next** or **Back** button causes the next search to start from the beginning of the cells in the table.

Application Browser (NFS 64370)

The table in the Application Browser now contains the following additional columns for each application.

- **Web App Type**, containing the type of Web application, if applicable; that is, **HTML Documents**, **Web Services**, **Rest Services**, or **JADE Forms**
- **Version**, containing application version number, if specified, to be displayed in the default About box for the application
- **Default Locale**, containing the default locale to be used when the application is run under a locale that is not supported by the schema of the application
- **Icon**, containing the icon, if specified, to be used for your application instead of the default icon
- **Font**, containing the typeface and attributes, if specified, of the font for the forms in the application to be used instead of the default font
- **Help File**, containing the name and location, if specified, of the application help file

Class Browser Displays the Schema Name (NFS 63547)

In earlier releases, the editor pane of the Class Browser included the schema name only if the class was a subschema copy class.

The editor pane now always includes the schema name in which the class is defined; for example:


```
Class: RootSchema::Form (189)
```

Collection Type Property Display (NFS 63360)

When a property name is clicked in the editor pane or the bubble help is shown for a property and the type of that property is a **Collection**, the displayed details now include the membership of the collection and the keys, if it is a **MemberKeyDictionary**; for example:

```
Name: allMyBigData (19)
Class: OdbcRoot ()
Type: DictBigData
        ------- DictBigData Collection Class Details -------
        Membership: BigData
        Member Keys:
              transactionDate (Date) ascending, case sensitive, sort order: Binary
Access: public
SubId: 18
Ordinal: 19
non-virtual

Inverses:
        myOdbcRoot of OdbcData
Update Mode: Automatic
Relationship Type: parent
```

Context-Sensitive Help to HTML5 Topics

Earlier releases of JADE provided context-sensitive help from the development environment to a Portable Document Format .**pdf** file, by default.

With the provision of the full product information library in both PDF (print) and HTML5 (Web) formats, you can now specify that context-sensitive help is obtained from .**htm** topics in the HTML5 Web format of the product information.

Context-sensitive help to HTML5 topics is controlled by a new **UseJadeWebHelp** parameter in the [JadeHelp] section of the JADE initialization file. This parameter is **true** by default, in which case it reads the new **JadeHelpBaseUrl** parameter. If a value is specified for the **JadeHelpBaseUrl** parameter, it uses that URL. If the value is **<default>** or it is empty, the URL is determined by the internal hard-coded URL for the current release. For example, the [JadeHelp] section of the JADE initialization file could contain the following parameter values.

```
[JadeHelp]
UseJadeWebHelp  = true
JadeHelpBaseUrl = https://www.jadeworld.com/docs/jade-2016/Default.htm
# Where the .htm extension is used, .html is also valid.
```

Set the value of the **UseJadeWebHelp** parameter to **false** if you want to continue using context-sensitive help to specific sections in the appropriate PDF files.

Displaying Breakpoints (NFS 63906)

The Debugger Breakpoints form, accessed from the **Breakpoints** command in the Browse menu, now displays breakpoint information in a table instead of a list box.

The table columns are:

1. Sequence number
2. Method in which the breakpoint is set
3. Line number of the breakpoint in the method
4. Line of source for the breakpoint (stripped of multiple spaces)

5. Source of any condition attached to the breakpoint
6. Pass count of when the breakpoint is to apply if the value is greater than zero (**0**)
7. Whether the breakpoint is disabled (**true**; otherwise empty)

In addition, the Breakpoints form in the JADE Debugger, displayed when you select the **Show All Breakpoints** command in the Breakpoints menu, now automatically sizes the table columns. The Breakpoints form is restored to its previous position and size if the **Save settings on exit** check box on the Debugger Options dialog is checked, but any previous column widths are no longer restored.

The Breakpoints table uses the **autoSize** property **AutoSize_ColumnMinimum** value so that the table adapts to whatever breakpoint information is currently active.

Editor Pane

This section describes the editor pane changes in this release.

AutoComplete Adding Parentheses (NFS 64288)

The **Editor Options** sheet of the Preferences dialog now provides the Auto Complete group box, which contains the **Use AutoComplete** check box that was in the Display Options group box in earlier releases.

When you have checked the **Use AutoComplete** check box in the Auto Complete group box, you can now check the **Insert Parentheses for Method with no parameters** check box if you want parentheses automatically inserted for a method that has no parameters.

The **Insert Parentheses for Method with no parameters** check box is unchecked by default. The opening and closing parentheses for a method call are automatically added to the text when the following are true.

1. The **Insert Parentheses for Method with no parameters** check box is checked.
2. A method name is selected from the auto complete list, by pressing a semicolon (**;**), dot (**.**), or space key.
3. The method does not have any parameters.

For example, if you type **customer.initi** and you select **initialized** from the auto complete list by pressing the **;** character, the result is:

```
customer.initialized();
```

AutoComplete List Box Entry Colors (NFS 64376)

The JADE AutoComplete list box entries are now displayed with the colors defined for the editor in your user profile (that is, on the **Editor** sheet of the Preference dialog) so that the color of entities in AutoComplete list box entries is the same that of the entity displayed in the method source in the editor pane.

The color options allow different colors to be shown for system types and for user types (specified on the **Window** sheet of the Preferences dialog). Note, however, that these colors are the same by default.

Displaying the Attribute Length in the Editor Pane (NFS 63564)

When using the F11 key to display details about the class under the caret in earlier releases, if you were looking for the length of an attribute, the attribute length was shown as the very last line of the text in the editor pane. In many cases, this went below the screen size and the bubble could not be scrolled to see the length (for example, if the attribute is used in a number of RPS mappings or as a key in a number of dictionaries).

When the details of a primitive type property are displayed in the editor pane, primitive type attribute text, including the property length, is now displayed before the lists of collections and RPS mappings in which the property is used.

Editor Pane Status Line (NFS 64119)

When you have checked the **View Status Line Info** check box on the **Editor Options** sheet of the Preferences dialog, the editor pane information in the status line of the background window now includes the number of characters selected within the editor pane.

The first field of the status line information now has the following format.

```
line:character:number-of-selected-characters
```

For example, **10:6:3** means line 10, character 6, and 3 selected characters.

This new field is displayed *only* when the **View Status Line Info** check box on the **Editor Options** sheet of the Preferences dialog is checked. (By default, this check box is unchecked.)

Folding Multiple-Line Block Comments (NFS 64118)

You can now fold multiple-line block comments (that is, comments bounded by the **/*** and ***/** notation as well as multiple consecutive line comments that start with double stroke character strings (**//**)) in the editor pane, by selecting the **Normal** or **Compact** folding option on the **Editor Options** sheet of the Preferences dialog. By default, folding is not enabled.

When folding in the editor pane is enabled, you can now expand and collapse multiple-line block comments. A minus icon (**-**) is displayed next to the first comment line that can be folded up to remove the majority of the comment from view.

JADE has implemented the following cases.

- A stroke asterisk (**/***) and asterisk stroke (***/**) comment block that stretches over more than one line, with the exception being a comment line started on the end of another line that can be folded for other reasons; for example:

  ```
  if ... then /*
  ```

- Multiple consecutive lines that start with double stroke character strings (**//**) comments where the **//** token is preceded only by spaces or tabs. If your folding option preference is set to:
  - **Normal**, the folding icon is displayed if three consecutive lines are in the format **white space// ...** (because folding the comments will show the first line underlined followed by the last comment line).
  - **Compact**, the folding icon is displayed if two consecutive lines are in the format **white space// ...** (because folding the comments will show only the first line underlined and not the last line).

Editor Clipboard Toolbar (NFS 63806)

The JADE development environment now provides an editor clipboard toolbar, which enhances the use of the internal JADE editor clipboards and the Windows clipboard. For details about using multiple JADE internal clipboards to copy the same code fragments (snippets) into many different unrelated classes where the code cannot be inherited from a common superclass and you have more than one code fragment, see "Using Multiple Clipboards when Refactoring Code" under "Editor Pane Shortcut Keys", in Chapter 2 of the *JADE Development Environment User's Guide*.

The Editor Clipboard toolbar has 11 buttons labeled **C** and numbers in the range **1** through **9** followed by zero (**0**), which represent the Windows clipboard status (the **C** button) and whether the internal clipboard of that number contains copied text. (The numbered buttons are in the sequence of the numeric keys on a keyboard.)

The button of a clipboard that contains no text is disabled and the letter or number is displayed in light gray, as shown in the following image.

If you copy text into a JADE internal clipboard buffer using the Ctrl+Alt+C key combinations, releasing the pressed keys, and then pressing one of the numeric keys that indicates the buffer (that is, **1** through **9**, and **0**), the button number is shown in blue font and in bold. The **C** buffer contains any text copied to the Windows clipboard.

To view the clipboard text in bubble help, move the mouse over the clipboard buffer.

Clicking a clipboard buffer button pastes the text into the current editor pane, replacing text that is currently selected (that is, it performs the same action as the Ctrl+Alt+V key combinations, releasing the pressed keys, and then pressing the numeric key that indicates the number of the buffer).

Notes The clipboard buffer buttons are disabled if the current schema is not a user schema (that is, it is the RootSchema) or if the editor pane does not have focus.

To hide the display of the editor clipboard toolbar or the floated Jade Clipboard Text Contents form, uncheck the **Show Clip Board Toolbar** check box on the **Window** sheet of the Preferences dialog or select the **Show Clipboard Toolbar** command in the View menu. If the editor clipboard toolbar is docked in the toolbar of the main development environment window, hiding the main development environment window toolbar also hides the editor clipboard toolbar.

When you exit from the JADE development environment, the contents of the internal JADE text editor clipboard buffers and any non-empty additional text entries defined in a floating toolbar that you have added (the **x<*n*>** entries) are saved in your user profile. Text in the Windows clipboard is not saved.

When you next initiate the JADE development environment, the JADE internal clipboard buffers and additional text entries are restored with the saved values.

A button in the toolbar is disabled if there is no text in that clipboard buffer.

If you change the contents of the Windows clipboard or an internal JADE editor clipboard, the appropriate text box is updated with that change.

If a JADE text editor pane has focus, clicking a button pastes that text into the editor pane, replacing any text that is currently selected.

For details about floating the toolbar and specifying additional text into the clipboard, see "Floating the Editor Clipboard" in the following section.

Floating the Editor Clipboard

When you float the editor clipboard toolbar by dragging the toolbar off the development environment main toolbar, the Jade Clipboard Text Contents form is displayed, containing the following in a table format.

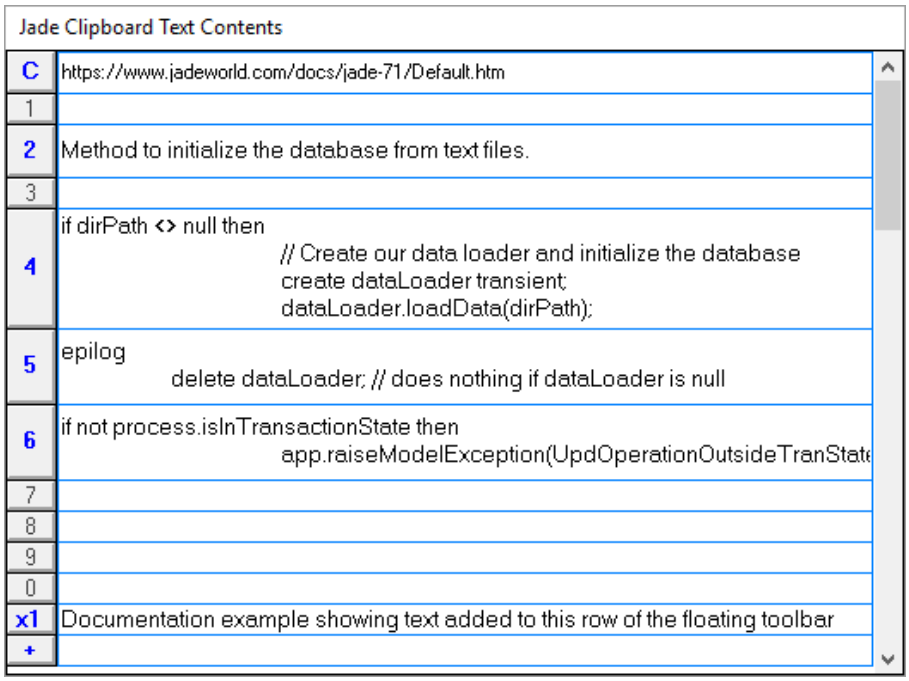
- A button labeled **C** (that is, Windows clipboard), followed by a text box that contains any text that is in the Windows clipboard.
- Buttons in the range **1** through **9** followed by zero (**0**), each followed by a text box that contains the text from each internal JADE text editor clipboard buffer, if applicable.
- A button labeled **+**, which enables you to add extra lines of text to the table view by typing the required text or code in the second column of the **+** row.

  There is no key sequence available to paste text selected in the editor pane into the **+** row. You must click in the second column of that row and specify the required text. Each additional row becomes an **x<*n*>** row when the cell loses focus.

A button in the table is disabled if there is no text in that clipboard buffer.

You can dock the editor clipboard toolbar separately at the left, top, right, or bottom of the JADE development environment background form. (See also "Editor Clipboard Toolbar (NFS 63806)", in the previous section.)

When the editor clipboard toolbar is first floated, each text box is sized vertically to match the clipboard contents up to a maximum of 150 pixels.



The dock or float status and the floating size and position of the editor clipboard are saved if you have set your **Save Windows** user preference to **true** (that is, you have checked the **Save Windows** check box on the **Exit** sheet of the Preferences dialog). When you next initiate the JADE development environment, the editor clipboard toolbar status is restored.

You can:

- Resize the table button entries vertically, which sizes the associated text box accordingly.
- Manually edit the text box contents, so that when the text box loses focus, the appropriate clipboard buffer is updated.

Note As the text boxes accept tabs, the standard tab action to leave a text box is not available. To change focus, you must use the mouse to click another window.

After text is entered into the field labeled **+** and the text box loses focus, the button is labeled **x<*n*>**, indicating that there is extra text in the buffer, with the **<*n*>** value being the extra sequence number. Another button labeled **+** with an empty text box is then created, allowing additional text to be available for pasting beyond the available clipboard buffers. (There is no key sequence available to paste the text, so you must click the associated button.)

Exposure Browser (NFS 63124)

The Exposure menu now provides the **Browse** command for C#, Web Services, and Web Consumer browsers, to display a browser similar to the Class Browser and which lists only the classes, properties, constants, and methods in the selected exposure.

When you select the **Browse** command, a hierarchy browser-type form displays:

- The class list in the upper left pane, containing a hierarchical list of classes in the exposure. Classes shown in:
  - Black are classes not in the exposure but are superclasses of the classes that are in the exposure
  - Green are included in the exposure but they have no property, constant, or method exposed

- Blue are those classes in the exposure that have at least one property, constant, or method exposed

Click on a class, to list the properties, constants, and methods exposed for that class, and to display the usual class information in the editor pane.

- The list of properties and constants in the upper middle pane that are exposed for the selected class. Exposed:
  - Properties are displayed using black text
  - Constants are displayed using green text, and they are listed after the properties names in the same list

  Click on a property or constant to display the usual browser information for the property or constant in the editor pane. For a C# exposure, the external name and type are also displayed.

  Right-click on a property to display a popup menu that contains the **References**, **Update References**, and **Read References** commands. Right-click on a constant to display a popup menu that contains the **References** command.

- The methods list in the upper right pane, containing the methods that are exposed for the selected class.

  Click on a method to display the method source in the editor pane. You cannot edit the source in this window.

  Right-click on a method to display a popup menu that contains the **References** command.

- An editor pane, containing a description of the currently selected entity. This text is read-only, and cannot be edited.

The displayed form has a View menu containing the **Show Composite View** command, which you can check or uncheck to toggle the composite view. This command is enabled only when the schema is versioned.

When the **Show Composite View** command is checked, the current schema is a versioned schema, and the selected schema is the *current* schema version, the properties, constants, and methods lists display entities that:

- Are versioned, including entries for both the current version and the latest version
- Have been deleted or removed from the exposure in the latest version
- Have been added to the exposure in the latest version

When the **Show Composite View** command is checked, the current schema is a versioned schema, and the selected schema is the *latest* schema version, the properties, constants, and methods lists display:

- Entities that are versioned, including entries for both the current version and the latest version

The added and deleted items are not displayed, because it is confusing as to which version of the schema the entries were deleted from or added to.

Notes The composite view uses standard color options that you control using the Preferences dialog.

The form does not offer the other display options supported by the hierarchy Class Browser or the options that are specified on the **Browser** sheet of the Preferences dialog.

When you press F4 to search for a class by name, the search results return only the classes in the exposure when this form has focus.

For an example of the Exposure Browser, see "Displaying a Hierarchical Web Service Exposure Browser", in Chapter 11 of the *JADE Developer's Reference*.

Extracting a Single Method (NFS 63590)

In earlier releases, you could extract a single method from a hierarchy browser by selecting a method in the Methods List of the Class Browser and then selecting the **Extract** command from the Methods menu.

You can now also extract a single method from the list of methods displayed after performing a references request of an entity, a global search, and other similar type requests. When a method is selected in a list of methods (for example, in the list showing the search results in the Global Find/Replace Results form), the Methods menu now includes the **Extract Method** command. The common Save As dialog is then displayed, to enable you to specify the name and location of your method file, as follows.

- The file name is constructed as *class-name_method-name.file-suffix*, defaulting to the name of the current type and method, with an **.mth** suffix (for example, when extracting the **Customer** class **getAddress** method, the default file name is **Customer_getAddress.mth**).
- The file suffix is as defined in the **Method** text box of the File Suffixes group box on the **Miscellaneous** sheet of the Preferences dialog.
- The directory to which the extracted method is saved defaults to your JADE working directory.

Find Type Dialog (NFS 63581, NFS 64124)

The Find Type dialog (accessed using the F4 function key) is now larger and can be resized.

In addition, when searching for a schema from the Schema Browser, the list box in the Find Schema dialog now includes the RootSchema.

Finding Unused Local Variables and Parameters (NFS 63969)

The finding of unused local variables and parameters from the Schema Browser, using the **Find Unused Local Variables/Parameters** command from the Schema menu, now enables you to display the results of the scan in a Methods Browser in a similar way to finding possible transient leaks functionality.

The Find Unused Local Variables and Parameters dialog now provides the Output Options group box, containing the following option buttons.

- **Print Preview**, which outputs the results that match your search criteria to the printer and display them in a print preview form. (In earlier releases, this was the only output option.)
- **Methods List Browser**, which displays each located unused local variable and parameter in a table on the Unused Local Variables/Parameters form (the default value).

The display includes the method name, the line number of the variable in the method of the variable, the name of the variable, and the type of the variable (that is, **variable** if it is a local variable, or **parameter**).

To display the method source with the unused item visible in the editor pane below the list on the form, click on a method in the list. You can edit this method source.

When a scan is performed using the **Find Unused Local Variables/Parameters** command from the Schema menu and the located unused variables and parameters are displayed in the Unused Local Variables/Parameters form, double-clicking a method entry in the list for an unused local variable entry results in *all* listed unused local variables being removed from the method source and the method is then recompiled. All of the unused local variable entries in the list for that method are removed from the list except for the last entry, so that you can view the method changes. You can also undo the changes, if required.

Notes If there is no remaining text between the **vars** and **begin** statements, the **vars** section is removed. No comments are removed.

You can also remove unused local variables by selecting the **Remove Unused Local Variables** command in the Methods menu.

If you click the only remaining entry for a method again and there are no changes required, the entry is removed from the list.

Global Find and Replace Handling (NFS 59252, NFS 63056)

The Global Search and Replace dialog now enables you to optionally search the names and values of all global constants and class constants. The new Search Entities group box contains the following check boxes.

- **Methods**, which searches all methods that match your specified search criteria for your specified text. This check box is checked by default.
- **Class Constants**, which when checked, also searches all class constant names and values for the specified string, applying the full word and case-sensitive options. The search will find the list of classes to examine using the same schema and class options that apply for a methods search. This check box is unchecked by default.

  When you check this check box, replace functionality is not available and the **Replace** button is disabled.
- **Global Constants**, which when checked, also searches all global constant names and values for the specified string, applying your search criteria including the full word and case-sensitive options. This check box is unchecked by default.

  The options in the Restrict Search To group box that control what classes to search are ignored when searching for global constant names and values. If the **Global Constants** check box is the only one checked in the Search Entities group box, the classes options group box is disabled.

  When you check this check box, replace functionality is not available and the **Replace** button is disabled.

Note If the **Keyword Search** check box is checked, the **Class Constants** and **Global Constants** check boxes are disabled.

When the search is complete, any entries that are located are displayed alphabetically, and depending on the entities that were included in your search, the list could contain the names of methods, class constants, and global constants. Clicking on a class constant or global constant entry displays the details of the constant definition.

When you use the Edit menu **Global Find/Replace** command or the Shift+Ctrl+F3 keys and JADE has located all occurrences of the text in the classes and schemas that match your search criteria, if the number of found occurrences within a method is greater than 1, the Global Find/Replace Results Browser now displays the number of occurrences of the searched text for a method in parentheses to the end of the method name; for example, **AllOrderedDoodahs::findCustomer (2)**.

JADE Debugger

This section describes the JADE Debugger changes in this release. (See also "Debugging Lock Exceptions (NFS 63339)" under "Locking", later in this document.)

Displaying Breakpoints (NFS 63906)

The Debugger Breakpoints form, accessed from the **Breakpoints** command in the Browse menu, now displays breakpoint information in a table instead of a list box.

The table columns are:

1. Sequence number
2. Method in which the breakpoint is set
3. Line number of the breakpoint in the method
4. Line of source for the breakpoint (stripped of multiple spaces)
5. Source of any condition attached to the breakpoint
6. Pass count of when the breakpoint is to apply if the value is greater than zero (**0**)
7. Whether the breakpoint is disabled (**true**; otherwise empty)

In addition, the Breakpoints form in the JADE Debugger, displayed when you select the **Show All Breakpoints** command in the Breakpoints menu, now automatically sizes the table columns. The Breakpoints form is restored to its previous position and size if the **Save settings on exit** check box on the Debugger Options dialog is checked, but any previous column widths are no longer restored.

The Breakpoints table uses the **autoSize** property **AutoSize_ColumnMinimum** value so that the table adapts to whatever breakpoint information is currently active.

JADE Debugger (PAR 46099)

After re-compiling a method while using the JADE debugger, the debugger continues to display the old method source under the following circumstances.

- The application is displaying a modal form.
- The debugger next stops on a breakpoint in the same method that was compiled and that method was the last displayed by the debugger.

When the application has stopped on a breakpoint and the **Continue** action is performed, the debugger clears the debugger display, including the last method shown when the execution stack becomes empty.

If a modal form is displayed, the execution stack does not become empty and the debugger displays are not cleared. If the next breakpoint encountered is in the same method, the debugger retains the currently displayed method.

When the application becomes idle and a modal form is displayed, the debugger clears the debugger displays and the new method source is displayed when a breakpoint is next encountered in that method.

Notes If the method is changed and re-compiled while the debugger has stopped on a breakpoint in that method (or in another method while that method is on the call stack), the interpreter will not reload that method logic until all copies of the method execution have exited from that method.

The debugger continues to display the old source while breakpoints are only encountered in that method until the application execution again becomes idle.

After a breakpoint is encountered in any other method, any breakpoint subsequently encountered in the changed method displays the new source, regardless of whether the old or new version of the method is being executed.

Implementing an Interface in an Imported Class (NFS 63989)

An imported class can now implement an interface. (In earlier releases, an imported class could not implement an interface, but could add methods to it.)

An imported class cannot implement an interface if it is already implemented by the exported class in the exporting schema.

Locating Unreferenced Methods (NFS 63113)

In earlier releases, methods reimplemented by a subclass within the same schema or another schema were shown as unreferenced.

Reimplemented methods are now considered unreferenced only if there are no references to the method and no references to any superclass implementations.

A method is considered unreferenced according to the following criteria.

- Not a condition
- Does not implement an interface
- Not used in an RPS mapping
- Not an event method

- Not used in a Web service
- Not exported
- Not a **JadeWebServices** class or subclass method
- Not a generated Web services method
- Not checked out
- Not an interface method with implementors
- Not a reimplemented RootSchema method
- Has no method references other than itself
- Is a reimplemented method but the superclass methods have no method references
- Not an application **initialize** or **finalize** method
- Not a JadeScript method
- Not a **create**, **delete**, **userNotification**, **sysNotification**, or **timerEvent** method

See also "Locating Unused Class Entities (NFS 63996)".

Locating Unused Class Entities (NFS 63996)

The Schema menu in the Schema Browser now contains the **Find Unused Class Entities** command, which enables you to locate unused classes, properties, class constants, and methods in one or more schemas.

The Find Unused Class Entities dialog enables you to select your search criteria (for example, the selected schema and all subschemas), your reporting options (that is, to include or exclude classes, constants, properties or methods), and your output option (that is, whether the results are displayed in a list browser or in print preview).

When you click the **Search** button, the selected schema, classes, and entities are searched for unused entities.

Note If the selected schema or schemas are versioned, only the classes in the current version of the schema or schemas are searched.

If you select the **Print Preview** output option, print-ready output is produced, containing the following columns.

- Schema
- Class
- Entity Type (the type of the entity that is unused)
- Class Entity Name (class constant, method, or property name; else blank when the unused entity is the class)

This list is ordered by schema name followed by class name and then unused constants, unused methods, and unused properties within each class.

When the default **List Browser** output option is selected, the output is displayed on the new Unused Class Entities form in a table that contains the following columns.

- Count (line number)
- Schema
- Class
- Properties

- Constants
- Methods

The **backColor** of the cell containing the name of the unused entity is drawn in light gray.

The table is ordered by schema name and class name and then unused constants, unused methods and properties within each class.

When you right-click on an unused entity in a cell in the table, the appropriate command for the Classes, Constants, Methods, or Properties menu for that unused entity is enabled. The commands that are available are:

- Classes
  - **References to *class-name***, which displays any references to the class
  - **Open Browser**, which opens a new Class Browser that displays and selects the class
  - **Remove From List**, which removes the selected row from the table
- Properties
  - **References to *property-name***, which displays any references to the property
  - **Open Browser**, which opens a new Class Browser that displays and selects the class and the property
  - **Remove From List**, which removes the selected row from the table
- Constants
  - **References to *constant-name***, which displays any references to the constant
  - **Open Browser**, which opens a new Class Browser that displays and selects the class and the constant
  - **Remove**, which removes the selected constant from the class
  - **Remove From List**, which removes the selected row from the table
- Methods, which displays a standard Methods menu except for:
  - Some commands are disabled (for example, the **New Jade Method** command, and so on)
  - The **Open Browser** command, which opens a new Class Browser that displays and selects the class and the method
  - **Remove From List**, which removes the selected row from the table

Note The class and property **Remove** commands are not available from the popup menu Unused Class Entities form, as you must remove these from the Class Browser because they potentially cause versioning.

An entity is considered unused according to the following criteria.

- Classes
  - Has no subclasses
  - Is not a subclass of **Application**, **Global**, or **WebSession**
  - Is not used in a Web services exposure
  - Is not the membership of a **Collection** class
  - Is not exported
  - Has no method references, excluding its own methods
  - Has no properties used as keys

- Is not used in an RPS mapping
- Has no property references in other classes
- Has no inverses

- Constants
  - Has no method references
  - Has no constant references
  - Is not exported
- Properties
  - Has no method references
  - Is not used as a key
  - Is not exported
  - Is not used in a Web service
  - Is not a form control property
  - Is not used in a condition
  - Is not an HTML property
- Methods
  - Is not a condition
  - Does not implement an interface
  - Is not used in an RPS mapping
  - Is not an event method
  - Is not used in a Web service
  - Is not exported
  - Is not a **JadeWebService** method
  - Is not a generated Web services method
  - Is not checked out
  - Is not an interface method with implementors
  - Is not a re-implemented RootSchema method
  - Has no method references other than itself
  - Is a re-implemented method but the supermethods have no method references
  - Is not an application **initialize** or **finalize** method

See also "Locating Unreferenced Methods (NFS 63113)".

Look and Feel

The JADE development environment has been refreshed to make it more approachable, usable, and productive. It has a more modern, flatter appearance to represent the JADE brand and iconography, with user preferences providing you with the ability to select small, medium, or large toolbar icons, for example.

Note Due to cut-off dates, the images throughout the product information library do not reflect the new look and feel. They will be updated in a subsequent release.

See also "Toolbar Icon Size (NFS 64381)", later in this document.

Nested Exception Limit when Reorganizing a Schema (PAR 63127)

In earlier releases, exception 4 (*Object not found*) could be raised, followed by a nested exception limit, when a loaded schema with source was reorganized and a **mouseMove** event occurred in the Class Browser. After a schema is reorganized, deleted, or unversioned, there is a period where notifications for the changes made are processed by the JADE development environment. During this period, the JADE development environment could have been unstable, because it was still referencing objects that no longer existed (objects replaced by a newer version, had been deleted, or unversioned). In addition, the development environment was paying a high processing cost in normal operation because it tried to handle the fact that objects could disappear in the middle of processing.

Changes have now been made that result in greater JADE development environment stability in a schema transition operation and that result in an improvement of the performance of normal development environment operations. These changes affect the following situations for any user of the JADE development environment or the JADE Painter:

- When a reorganization is about to start the final schema transition phase (where the old schema version is deleted and the new version instantiated). This phase is after all instances in files have been reorganized.
- When a schema is deleted and the schema is about to be removed from JADE (after all instances have been deleted).
- A schema is unversioned and the versioned schema is about to be deleted.

In the above situations:

1. Each JADE development environment or Painter process is disabled and a modal dialog is displayed with the following message.

   ```
   The use of the JADE IDE has been suspended until a schema transition phase is
   completed.

   This is because of the instantiation of a schema version, a schema delete or a
   schema unversion.

   Normal operation should resume shortly.
   ```

   You cannot interact with the JADE development environment during this time.
2. The schema transition is completed, which usually takes just a few seconds.
3. Each JADE development environment or JADE Painter session is informed that the schema transition has completed.
4. Each form affected by the schema transition is updated to refresh the affected objects.
5. The modal form is unloaded and you can again interact with the JADE development environment or Painter.

   Note Notifications about the schema transition can still be received after the development environment is available again, but the JADE development environment is now up-to-date and should not be significantly affected.

The result is that each JADE development environment or Painter process is stable during the schema transition and the development environment performance has improved.

Painter

This section describes the JADE Painter changes in this release.

Hierarchical List of all Controls Painted on the Active Form (NFS 63591)

The JADE Painter now provides a form that displays a hierarchical list of all controls painted on the currently active form; for example, if you want to inspect the controls painted on a complex form. Activate the form by selecting the **Show Control Hierarchy Dialog** command from the Window menu of the JADE Painter or by pressing F5 when the Painter has focus.

A list box displays a hierarchical list of parent-child relationships of each control painted on the form. Each control is listed alphabetically by control name under its control parent entry. Each entry is displayed as *control-name* (*control-class-name*) '*caption*'; for example:

```
btnEdit (Button) 'Edit'
```

The caption value is displayed only if a **caption** property value has been defined for the control and it is not null.

Each entry that has children is displayed in blue text, while each leaf child entry is shown in black text.

The Windows system font is used to display the entries so that any caption text in other languages can be displayed.

Each time you select a control in the painted form, that entry is selected in the Hierarchy for Form dialog. The hierarchy is expanded, if required, and the entry is centered in the list box, if possible.

The list is updated whenever a control is added to or deleted from the painted form, if the type of a control is changed, the parent of a control is changed, a caption is changed, or if another form being painted is activated.

Clicking on a control entry in the list box on the Hierarchy for Form dialog selects that entry in the Painter and the Painter is activated if it is not already active. In addition:

- Pressing F4 when the Hierarchy for Form dialog has focus causes the Properties dialog to be displayed.
- Double-clicking a control name in the list box of the Hierarchy for Form dialog causes the Properties dialog to be displayed after the control is selected in the Painter.
- Pressing F5 when the Properties dialog has focus displays the Hierarchy for Form dialog.

The Hierarchy for Form dialog is unloaded if the currently selected form being painted is unloaded or if the JADE Painter is closed.

Click the **Stay on top of Painter** icon at the top left of the dialog or select the **Control Hierarchy on Top** command from the Options menu to keep the Hierarchy for Form dialog on top of the Painter. Conversely, repeating these actions toggles the pinning of the dialog on top of the Painter and the check status of the menu command.

The window state and size of the dialog and the checked status is saved when you close the dialog. When the dialog is next displayed, the state of the form that is displayed is determined by the last saved state.

Selecting a Skin for a Painted Form (NFS 64262)

In earlier releases, you could not verify a form being painted in the development environment JADE Painter against the skin to be used at run time.

The JADE Painter now enables you to select a runtime skin that is used to display any form that you are painting, by selecting the **Select Skin** command from the File menu. The Select or Cancel a Skin form is then displayed, to enable you to select the runtime skin in the **Choose Skin** combo box. If you have not loaded any runtime skins into your JADE system, the default value of **<None>** is the only value available in this combo box.

Tip The **examples\skins** subfolder of the JADE install files contains runtime skins that you can load. For details about loading the **SampleSkins.ddb** file, see the **readme.txt** file in that subfolder.

When you select a runtime skin, the **Control Examples** pane on the form displays an example of controls (and menu and menu items, if selected for display) using that skin.

When you are happy with the controls and menu on the painted form displayed in that skin, click the **Apply** button. That skin is then applied to any forms being painted. If a skin is selected, the JADE Painter caption reads JADE Painter : *schema-name::form-name* - using skin '*skin-name*' - [*caption-of-form*]; for example:

```
JADE Painter : DemoSchema::Company - using skin 'Windows Broadbean' - [Company]
```

In addition, any subsequent forms opened in the JADE Painter are displayed using the selected runtime skin.

The selected skin is saved in your user preferences when you close the JADE Painter and restored when you re-open the Painter.

Resizing Dialogs (PAR 64258)

To cater for larger class and method names, the following dialogs can now be resized.

- Currency Format
- Long Date Format
- Short Date Format
- Number Format
- Time Format
- Copy Method As

Reusing Forms that Display Method Source (NFS 63234)

You can now specify whether you want to reuse the same form to display the source of a method, when possible, rather than creating a new form for each such request. (In earlier releases, when the editor pane had focus and you pressed F12, JADE created a separate form to display the method for each such request.)

The option to reuse the same method source window is used when:

- A method source window is restored when the JADE development environment is initiated.
- You double-click an **initialize** or **finalize** method entry on the Application Browser.
- You double-click on a method in a Class Browser.
- You double-click on a method in the Methods List of a Class Browser.
- You press F12 when in a method source window.
- You press F11 when a method identifier is selected in a method.
- You double-click on an entry in a breakpoint list browser.
- You double-click on a method in a Library List browser.
- You copy a method using the Methods menu **Copy** command.
- You add a new method while the method source window has focus.

The option to reuse the same form is controlled by the new **Reuse Same Method Source Window** check box on the **Method** sheet of the Preferences dialog.

In addition, the View menu from the Class Browser now provides the **Use Same Methods Source Window** command, which toggles the reuse of the same method source window. This command controls only any request made from the *current* editor pane, so you can use it to determine whether the **Reuse Same Method Source Window** check box on the **Method** sheet of the Preferences dialog is checked.

When you have specified that you want to reuse the same form to display method source and you press F12 in the editor pane, JADE initially creates a new form to display the method source, and a list box that displays the history of how you got to that form. Each subsequent request to show another method uses the same window and adds the additional navigational history to the list box.

Click on an entry in the list box to display the method source for that entry, unless the currently displayed method has been changed, in which case the Save Method message box prompts you to save and compile your changes, to discard them, or continue editing the method. Conversely, if the method source in a shared window has been changed, when the new method source request is made, a message box is displayed informing you that the window cannot be shared (until the changes are saved or restored), and asking whether you want a new method source window to be opened. You can then cancel the request or click **OK** to open another window.

Note The behavior of the F12 function key has been changed, so that if the current method has been changed, the development environment now prompts you to save and compile your changes, to discard them, or continue editing the method before the new method source window is displayed. This avoids two changed separate copies of the same method being open, with potentially different sources, leading to confusing results when moving from form to form.

If you close a form that is displayed in the method source browsing history list box, that entry is removed from the list box when the form is next activated. Similarly, if a method that is displayed in the method source browsing history list box is deleted, that entry is removed from the list box.

The method source browsing history list box entries are updated to use the current form caption when the form is activated.

Notes The history list can contain both versioned and non-versioned methods. The appropriate title and skin are displayed when each method is made current.

Only the currently selected method is restored after your work session is closed and then restarted and the option to restore windows is enabled. (The history is *not* restored.)

Saving Text in a Workspace or the Editor Pane (PAR 64087)

In earlier releases, the File menu **Save** command and the **Save** toolbar button were inconsistently enabled and disabled; for example, they were always enabled when a method was displayed, even though it was unchanged.

The File menu **Save** command and the **Save** toolbar button are now enabled only after the text in the editor pane or a Workspace has been changed. After the saving or compiling of a method, the command and toolbar button are disabled until the next change is made to the text.

In addition, the JADE development environment File menu **Save As** command is now displayed and enabled only when a Workspace editor pane has focus, because it applies only to saving a Workspace file.

Scaling the Splash Screen (PAR 63654)

In earlier releases, the JADE splash screen was not scaled under different monitor dots per inch (dpi) settings, so that the text drawn for different items on the screen could overlap and was not presented well.

The **StatusPos**, **ApplicationPos**, **SchemaPos**, **ServerPos**, **PathPos**, **VersionPos**, and **AviPos** splash screen parameter values in the [Jade] section of the JADE initialization file are now based on 96 dpi. The splash screen positions are scaled if the user has a different dpi setting.

Script Management (NFS 63850)

JADE now enables you to organize and manage your scripts, to facilitate their reuse and to reduce duplication. You can now:

- Subclass the **JadeScript** class in your user schemas. You can run and debug a JADE script from any subclass of the **JadeScript** class.

Running a JADE script in a subclass of **JadeScript** using the **jadclient** non-GUI application requires the class to be specified in the **executeClass** command line parameter.

- Debug a JADE Workspace.

  In the Workspace window, select the text to be executed and then select the **Debug** command from the Jade menu. (If you do not select some text, the entire contents of the Workspace window are debugged.) The selected text is compiled as a method and executed in debug mode.

  Note You cannot set breakpoints in the original Workspace window.

  The debug execution initially stops on the first line of execution. You can use the JADE Debugger to set debug breakpoints in the method. You can then step through the logic or debug and run it to meet your requirements by using the debugger.

Swapping Accelerator Keys

This section describes the ability to swap F5 and F9 accelerator keys and F11 and F12 accelerator keys in this release.

Swapping the F5 and F9 Keys (NFS 63079)

Visual Studio uses the F9 key to set breakpoints, by default, and uses the F5 key to run and continue debug execution. JADE uses the F5 key to set breakpoints and the F9 key to run and continue debug execution. Switching between the two development environments can lead to confusion and frustration when using the F5 and F9 keys.

The **Editor Key Bindings** sheet on the Preferences dialog now provides the **Swap F5 (Toggle Breakpoint) and F9 (Execute/Continue) accelerators** check box. As this check box is unchecked by default, the behavior in earlier JADE releases is unchanged; that is, the F5 accelerator key causes break points to be toggled and the F9 accelerator key executes or debugs a script method and continues execution of the JADE debugger.

If you check this check box, the menu caption and accelerator key for the Jade debugger Breakpoints menu **Toggle** command is changed to be triggered by the F9 key. In addition, the JADE development environment Jade menu **Execute**, **Debug**, **Unit Test**, and **Unit Test Debug** commands and the JADE debugger Debug menu **Continue** and **Animate** commands will be changed to use the F5 key.

Notes The setting of the **Swap F5 (Toggle Breakpoint) and F9 (Execute/Continue) accelerators** check box is saved when you extract your user preferences to a file and it is restored when you reload your preferences.

Changing the setting of the check box takes effect in the JADE development environment as soon as you close the Preferences dialog. It will not take effect within any currently active JADE debug session, and it is read only when you initiate a new debug session.

Swapping the F11 and F12 Keys in the Editor Pane (NFS 63323)

The **Editor Key Bindings** sheet on the Preferences dialog now provides the **Swap F11 (Show Symbol/Open class browser) and F12 (New Window/Bookmark) accelerators** check box. You could swap the F11 and F12 accelerator key functionality, for example, if you want the F12 key in the JADE editor pane to have a similar meaning to F12 in Visual Studio, where the definition of the selected symbol is displayed.

Note Swapping the F11 and F12 accelerator keys in the editor pane does not affect the interpretation of the F11 function key in the JADE Debugger.

When this option is **false** (the default) while in the editor pane of a method, when pressing:

- F11, information for the currently selected symbol is displayed. If the symbol is a method, a new method window is opened for the method.
- F11+Shift, a new Class Browser is opened if the symbol is a class.

- F12, a new editor pane is opened for the currently displayed method.
- F12+Shift, the current state is saved as a bookmark or the currently defined bookmark state is restored.

As this check box is unchecked by default, the behavior in earlier JADE releases is unchanged.

If you check this check box, the F11 and F12 accelerator key functionality is swapped.

Notes The setting of the **Swap F11 (Show Symbol/Open class browser) and F12 (New Window/Bookmark) accelerators** check box is saved when you extract your user preferences to a file and it is restored when you reload your preferences.

Changing the setting of the check box takes effect in the JADE development environment as soon as you close the Preferences dialog. It will not take effect within any currently active JADE session.

Toolbar Icon Size (NFS 64381)

You can now select the size of icons on JADE development environment toolbars. The size options are **Small** (16x16 pixels), **Medium** (32x32 pixels), and **Large** (48x48 pixels). You can select the toolbar icon size from the:

- Hierarchy Browser View menu **Icon Sizes** command, which displays the submenu containing the **Small Icons**, **Medium Icons**, and **Large Icons** commands. A checkmark indicates the size that is currently selected.
- Toolbar Icon Sizes group box on the **Browser** sheet of the Preferences dialog, which contains the **Small**, **Medium**, and **Large** option buttons.

Any change to this option is retained in your profile data and restored the next time you initiate the JADE development environment. It is also saved and restored when you export and reload your user preferences (from the **Miscellaneous** sheet of the Preferences dialog).

The toolbar icon size affects the following toolbar icons.

- Development environment toolbar
- Painter toolbars
- Painter Properties dialog
- Painter Hierarchy for Form dialog
- Debugger toolbar

Changing the icon size causes the new icon size to be immediately applied to all open relevant windows except for the JADE Debugger, where the option is loaded only at debugger initiation.

Type Methods (NFS 63225)

JADE now supports *type methods*, which provide a way of calling a method declared on a type (class, primitive, or interface) without having to have an instance of the type. While the method is declared on a type and has the same scope as an instance method, at run time the receiver is the type on which the method is declared; *not* an instance of the type.

Type methods enable you to define and call methods on class, primitive, or interface types without having to instantiate a dummy object, which simplifies aspects of JADE application development. Examples of a type method are a:

- Factory method that creates and initializes instances of a class.
- Method that sums the values of all of the properties for all instances of a class (for example, a **Product** class could have a type method to compute the average price of all products).

A type method is identified by the new **typeMethod** method option on the method declaration; for example:

```
demoTypeMethod() typeMethod;
vars
begin
end;
```

Note As the receiver is not an instance of the type, you cannot use the **self** system variable in a type method, but you can use the new **selfType** system variable, which references the type. You can use the **selfType** system variable in both type methods and existing class methods.

Two syntaxes are supported for calling a type method. To avoid overloading the dot operator (**.**) notation, the new **@** operator identifies that a type method is being called.

- The following syntax uses the name of the type to call a type method using the name of the type.

  The following example is a call to the **demoTypeMethod** type method in class **C1**.

  ```
  C1@demoTypeMethod();
  ```

  In this example, the **@** operator notation makes it clear that the method being called is a type method defined on the type **C1**; not on the **Class** class, as would be the case with the **.** operator.

- The following syntax calls a type method using a variable, parameter, or property.

  The following example is a call to the **demoTypeMethod** type method on class **C1**, using a local variable of type **C1**.

  ```
  vars
      c1 : C1;
  begin
      c1 := C1.firstInstance();
      c1@demoTypeMethod();
  end;
  ```

  In this example, the method can be called even if the value of **c1** is null. The variable is used by the compiler to determine the method to call; in this case the **demoTypeMethod** type method declared on the **C1** class.

  Note Because **demoTypeMethod** is a type method, the receiver is not the value of **c1** but the *type* of the value of **c1**.

See also "Invoking a Type Method", in the following section.

The Jade Method Definition dialog or the External Method Definition dialog now contains the **Type Method** check box, which is unchecked by default. You can declare an existing method as a type method, by adding the **typeMethod** method option to the method signature and then recompiling the method. (Conversely, convert a type method to a class (instance) method, by removing the **typeMethod** modifier, removing any **selfType** system variable or changing the **selfType** system variable to **self**, and then recompiling the method.)

Note Changing to or from an instance method from or to a type method requires recompiling all referencing methods, which will fail if the wrong operator (the **.** or **@** notation) is used in the changed method.

The Methods List of browser windows now has three folders: **All**, **Instance**, and **Type**. By default, the **All** folder is displayed; that is, all instance methods and all type methods are displayed. By default, instance methods are displayed in a black font and type methods are displayed in a dark blue font in the Methods List of browser windows.

You can reimplement type methods on subclasses. As is the case for instance methods, the signature must be the same in all implementations. At run time, the value of the variable is used to determine which implementation of **demoTypeMethod** will be called. If the value of the variable is an instance of a subclass and a type method reimplements a type method on the class, the reimplemented type method will be called. If the value of the variable is null, the type method on the declared type of the variable is called.

Transient type methods (created by the **Process** class **createTransientMethod** method) are supported.

A type method can...	A type method cannot...
Be a Jade method or an external method.	Use **self** or any instance methods or properties.
Be defined on a JADE class, a primitive type (for example, **Integer**), or an interface type.	Be a constructor or a destructor.
Be used with the **public**, **updating**, **protected**, **serverExecution**, and **clientExecution** method options.	Be a notification or an event method.
Reimplement a type method with the same signature defined on a superclass but it cannot reimplement an instance method (and the reverse).	Be abstract or a condition.
Invoke its superclass implementation using the **inheritMethod** instruction or the C++ **jomInheritMethod** Application Programming Interface (API). See also "Type Method API Calls", later in this document.	Be combined with the **lockReceiver**, **updating**, **mapping**, or **webService** method option.
Be declared on a subschema copy class, with the usual JADE method visibility applying (that is, a subschema copy method cannot be accessed by applications running in a superschema or a peer schema if the root type or a subschema copy for the class already exists in the schema or a superschema).	Reimplement a method defined on a subschema copy class in a superschema or on the root class if the type method is defined on a subschema copy class.
Be imported from a package.	Be **conditionSafe**.
Be defined on an imported class.	Be used in a **unitTest** method.
	Be called from a REST service.

Invoking a Type Method

You can call a type method by qualifying the method call with the name of the class to type, as shown in the following code fragment, which specifies the **someMethod** method on the **Company** class.

```
Company@someMethod();
```

Call a type method exported from a package in the same way, as shown in the following code fragments.

```
ImportedCompany@someMethod();

Package::Company@someMethod();
```

You can provide an application context for the call to a type method, as shown in the following code fragment.

```
Company@someMethod() in someApplicationContext;
```

You can also call a type method by using a variable, parameter, or property. In this case, the type of the value of the variable identifies the type method to call.

You cannot call a type method indirectly using the **Object** class **sendMsg** or **sendMsgWithParams** method, nor can you explicitly call an exported type method using the **Object** class **invokeMethod** method. Illegal calls to a type method result in the new error 1185 *(Invalid call to a type method)* to occur.

The **Object** class now provides the **sendTypeMsg**, **sendTypeMsgWithParams**, and **sendTypeMsgWithIOParams** methods. Calling an instance method with one of these methods results in the new error 1184 (*The requested method is not a type method*) to occur.

Type Method API Calls

In addition to the JADE Application Programming Interface (API) calls documented in Chapter 3 of the *JADE Object Manager Guide*, JADE now implements the following new APIs to call a type method on a class and primitive type.

The **jomSendTypeMsg** API call sends a message to the type of the receiver, which is the root type instance of the class identified by the **ClassNumber** parameter. If a type method is defined on an imported class, the receiver is the class instance of the exported class (*not* the imported class instance).

```
int JOMAPI jomSendTypeMsg(const DskHandle *pHandle,
                          ClassNumber     classNumber,
                          DskParam        *pMessage,
                          DskParam        *pParams,
                          DskParam        *pReturn,
                          LineNumber      lineNo);
```

The **jomCallPrimTypeMethod** API call invokes a primitive type; for example, from an external method.

```
int JOMAPI jomCallPrimTypeMethod(const DskHandle *pHandle,
                                 TypeNum         primitiveNumber,
                                 DskParam        *pMessage,
                                 DskParam        *pParams,
                                 DskParam        *pReturn,
                                 LineNumber      lineNo)
```

The receiver of a **jomCallPrimTypeMethod** API call to a primitive type method is the root type instance of the primitive type specified in the **TypeNum** parameter.

Instance methods cannot be called by the **jomSendTypeMsg** or **jomCallPrimTypeMethod** API. An attempt to do so results in exception 1184 (*The requested method is not a type method*) being raised.

Unit Testing and Code Coverage

This section describes the unit testing and code coverage changes in this release.

Automatically Running Changed Test Methods (NFS 64110)

In earlier releases, you could not automatically run modified or newly added test methods in the Unit Test Runner application. This could take a lot of time and effort writing tests and having to keep swapping to the Unit Test Runner form to run the tests.

The **Select Tests** list box at the left of the Unit Test Runner form now includes the status of all selected tests. The display indicates whether a test has passed (green tick), failed (red cross), or has not been run (blue question mark). The display is updated as new unit tests are added to the class (the status will be set to not run), compiled (the status is set to not run), or deleted (the test is removed from the list).

In addition, the new View menu provides the following commands that enable you to select methods based on their current status.

- Include Passed Tests in Results, which is checked by default, and was in the File menu in earlier releases
- Select Passed Tests
- Select Failed Tests
- Select Not Run Tests
- Refresh (F5), which was in the File menu in earlier releases

For example, if a number of tests are added to the unit test or existing tests are recompiled, you can select all tests that have not been run (by selecting and this checking the **Select Not Run Tests** command). You can then run these tests by clicking the **Run** button in the lower area of the form. The **Progress** and **Results** tables are then updated with the results from each test run, and the list of selected tests reflects the status of tests over all runs.

This works within the context of the Unit Test Runner; for example, if the test is run against a test class, only modified tests within that class are executed, or if you run the test against a schema, only tests visible to that schema are executed.

When focus is in the **Results** pane, the Popup menu provides the **Copy Results to Clipboard** command.

The following two error messages can now occur.

- 1466 (*Unable to run a unit test*)
- 1467 (*Unit tests failed*)

When unit tests are run in batch mode (that is, in **jadclient**), the exit code is now zero (**0**) if there were no errors; otherwise an exception code. For example, if one or more tests fail, the exit code is now 1467, in which case you can check the **JadeUnitTest.log** file for details of the failed test or test. Error 1466 is an indication that an incorrect parameter was specified; for example, an invalid schema name or test name, and so on. (In earlier releases, the exit code was the number of failed tests.)

Enabling and Disabling Code Coverage within Unit Tests (NFS 64109)

In earlier releases, code coverage had to be coded (by creating an instance of the **JadeTestRunner** class and using the **runTests** method or manually enabled by selecting the **Start** or **Stop** command from the **Code Coverage** command in the Jade User Interrupt submenu (accessed from the system tray at the right of the Taskbar).

You can now start or stop code coverage from the Unit Test Runner form, whose File menu now includes the following commands that enable or disable, report, and view code coverage.

- Code Coverage
- Report Code Coverage
- View Code Coverage

In addition, you can now also enable or disable code coverage in a batch mode (non-GUI client) application, by specifying the optional **codeCoverage=true|false** parameter before the **endJade** parameter on the command line.

Unit Test Forms Resizing (NFS 63663)

In earlier releases, the Unit Test Runner form and the Call Stack Browser size and position were not retained.

The window state for the current user of the Unit Test Runner form and the Call Stack Browser are now saved and restored when used as part of the JADE Unit Test framework.

JADE Initialization File

This section describes the JADE initialization file changes in this release. (See also "Debugging Lock Exceptions (NFS 63339)" for details about the **DefaultProcessSaveLockCallStack** parameter that can be specified in the [JadeClient] and [JadeServer] sections, or "JADE Inspector (NFS 63539, NFS 34896)" for details about the **UseSameWindow** parameter in the [JadeInspector] section.)

Cache Size Limit Default Values (PAR 62677)

Cache size limit default values increased in JADE 7.0.05 but were not updated in the product information prior to the to the JADE 7.0.12 and JADE 7.1.05 releases.

In the JADE initialization file, the default value of the:

- **ObjectCacheSizeLimit** parameter in the [JadeClient] and [JadeServer] sections has increased from 8M to 80M
- **TransientCacheSizeLimit** parameter in the [JadeClient] and [JadeServer] sections has increased from 5M to 40M
- **RemoteTransientCacheSizeLimit** parameter in the [JadeServer] section has increased from 5M to 40M

IndexLoadFactor Parameter in the [PersistentDb] Section (PAR 64067)

The [PersistentDb] section of the JADE initialization file now contains the **IndexLoadFactor** parameter, which specifies the percentage of entries that are retained when an index block becomes full and is split. A value of **95** means that 95 percent of entries are retained in the current block and 5 percent of entries are moved to a new block.

Statistically, a 66 percent load factor (the default value) provides optimal loading when entries are added in random key order and a higher load factor (for example, 95 percent) provides better loading when entries are added in sequential key order.

The minimum parameter value is **50**, and the maximum parameter value is **95**.

Thread Parameters in the [JadeServer] Section (PAR 62498)

In earlier releases, when any of the new **MinShortThreads**, **MaxShortThreads**, **MinLongThreads**, or **MaxLongThreads** parameters were not specified in the [JadeServer] section of the JADE initialization file, server initialization wrote them out with the corresponding **MinServerThreads** or **MaxServerThreads** parameter value carried over, or with the default **MinServerThreads** or **MaxServerThreads** parameter value.

From this release, when the **MinServerThreads** and **MaxServerThreads** parameters are not specified to seed the **MinShortThreads**, **MaxShortThreads**, **MinLongThreads**, and **MaxLongThreads** parameter specifications, the **MinShortThreads**, **MaxShortThreads**, **MinLongThreads**, and **MaxLongThreads** parameters now use their own default values, which are as follows.

- **MinShortThreads** default value is 20
- **MaxShortThreads** default value is 100
- **MinLongThreads** default value is 15
- **MaxLongThreads** default value is 100

If the value of a **MinServerThreads** or **MaxServerThreads** parameter is used to seed any of the **MinShortThreads**, **MaxShortThreads**, **MinLongThreads**, and **MaxLongThreads** parameters during initialization, the **MinServerThreads** and **MaxServerThreads** parameters are now removed from the [JadeServer] section of the JADE initialization file after initialization, as they are superseded by the new parameters.

When the **MinServerThreads** and **MaxServerThreads** parameters are not specified in the [JadeServer] section and the new **MinShortThreads**, **MaxShortThreads**, **MinLongThreads**, and **MaxLongThreads** parameters do not exist, the new parameters are written to the [JadeServer] section with the value **<default>**.

ThreadPriority Parameter Removal (PAR 63251)

The **ThreadPriority** parameter has been removed from the [JadeClient] and [JadeServer] sections of the JADE initialization file.

Because JADE was incorrectly overriding a below-normal thread priority, this parameter has been removed on client and server nodes, as standard management tools exist to set the default thread priority.

JADE Inspector (NFS 63539, NFS 34896)

The JADE Inspector form has changed as follows.

- When the **Object** class **inspectModal** method is called, the initial form is shown modally. When additional forms are opened by double-clicking an object in a displayed Inspector form, the opened forms are now displayed as children of the initial modal Inspector form. These forms always sit on top of the initial modal form, and you can access all of the Inspector forms.

  Closing the initial modal form also closes all of its inspector child forms.

  In earlier releases, each form was shown modally and you could access the top form only. To access the prior form displayed, the top form had to be closed.

- The Options menu now provides the **Use Same Window** command, which is unchecked by default. When the **Use Same Window** command is unchecked, each double-click of an object in an Inspector form opens a new Inspector form; that is, the behavior in earlier releases is unchanged.

  Clicking the **Use Same Window** command toggles whether the menu item is checked (that is, whether the form is used). Toggling the value of this command writes the **true** or **false** value to the new **UseSameWindow** parameter in the [JadeInspector] section of the JADE initialization file.

  When the **Use Same Window** command is checked, each double-click of an object in an Inspector form re-uses the same form to display the selected object, replacing the previously displayed object. In addition, a new pane is displayed on the right of the form, showing a hierarchical list box displaying all of the objects that have previously been inspected. The hierarchy indicates the history of how the objects were inspected. The entries display the value of the **name** property if it exists in the object, followed by the class name and the JADE object identifier (oid). Clicking on an entry in the historical list displays the selected object again.

- The Inspector form now provides the History menu, which displays up to the last 25 objects inspected, in the reverse order that they were double-clicked. Click an entry in the hierarchical history list at the right of the form, to re-display that object.

  Note The History menu list is not affected by clicking an item in the list or by clicking an item in the hierarchy history list box.

  The History menu displays a check mark at the left of the object currently displayed in the Inspector form (if it is in the menu item list). Press the Alt+↑ (up arrow) or Alt+↓ (down arrow) accelerator keys to step to the previous or next entry, respectively, in the order in which they were originally displayed.

- The File menu is now always displayed. In earlier releases, this menu was hidden if the form was displayed modally.

  The File menu always contains the **Exit** command and it contains the **Close all** command if the JADE Inspector form was not shown modally.

  In addition, the File menu now provides the **New Window** command, which opens a new Inspector form for the currently displayed object (regardless of whether the **Use Same Window** command in the Options menu is checked). To display an object in a separate form that can be retained regardless of what objects are inspected in the original form or forms, check the **New Window** command.

- When a collection is displayed on the left of the JADE Inspector form, clicking on another collection entry now preserves the vertical scroll position of the text box that displays the property values on the right, if that position exists.

JADE Interpreter Output Viewer (NFS 46405)

In earlier releases, the JADE Interpreter Output Viewer enabled you to copy the text, save the text to a file, add an annotation, and change some display settings.

The JADE Interpreter Output Viewer has been enhanced to include:

- Print setup
- Ability to print all text or selected text
- Copy, cut, delete, and paste functionality
- Ability to find text in the display, searching forwards or backwards, case-sensitive or not, as well as searching from the top, the bottom, from the current cursor position, and using the current selection
- Specifying whether word-wrap is on or off for the displayed text
- Toggling the pausing and resuming of the update display, so that **write** statements continue to be output while you are attempting to interact with the text without the new text interfering with your interaction
- The accumulated output has increased from 1,000 lines to 4M bytes of text, truncated to the nearest full line

For details about using the JADE Interpreter Output Viewer, see "Using the Jade Interpreter Output Viewer", in Chapter 1 of the *JADE Runtime Application User's Guide*.

JADE Logical Certifier Utility

This section describes the JADE Logical Certifier utility changes in this release.

JADE Logical Certifier Error 33 (PAR 62880)

The JADE Logical Certifier utility now detects error **33** *(DynaDictionary incomplete or inconsistent*) when a **DynaDictionary** instance references a missing or inconsistent class or property definition; for example, when the membership class of the **DynaDictionary** is deleted, or when a property that is used as a member key property of a **DynaDictionary** is deleted. The repair action is:

```
delete oid
```

Logically Certifying Meta Data (PAR 62689)

In earlier releases, logically certifying meta data did not report invalid **VersionInfo** instances.

The logical certification of meta data now reports as errors versioned meta schema instances where the schema is invalid or it is not versioned. Error 99 is reported if any such instances are encountered (no fix is generated for this error).

Note This new check may encounter old errors in your user systems. For assistance with creating manual fixes to correct invalid versioned meta schema instances, contact JADE Support (jadesupport@jadeworld.com).

Orphan Block Checking (PAR 63378)

The Logical Certifier now handles deleting collections that have missing blocks.

Orphan blocks may be created when a dictionary with missing blocks is deleted using a **delete** or an **orphan** repair. If any unreachable blocks exist prior to the collection being deleted, they are not reported as orphans until after the collection header is deleted. A message is logged to the Logical Certify **repair.log** file if a collection that contains missing blocks is deleted. You should run the Logical Certifier again, to determine whether any orphan collection blocks have been created.

The Logical Certifier now provides the following error codes for the orphan checks.

- Error 40 - Orphan dictionary block

  This error is detected when a dictionary block is found but the parent instance that owns the dictionary does not exist.

Repair:

```
orphanBlock filename
```

- Error 41 - Orphan blob/slob

  This error is detected when a blob or slob subobject is found but the parent instance that owns the blob or slob does not exist.

  Repair:

  ```
  orphanSlobOrBlob filename
  ```

- Error 42 - Orphan dynamic property cluster

  This error is detected when a dynamic property cluster is found but the parent instance that owns the cluster does not exist.

  Repair:

  ```
  orphanCluster filename
  ```

- Error 43 - Orphan subobject (collection)

  This error is detected when a collection subobject is found but the parent instance that owns the collection does not exist.

  Repair:

  ```
  orphan oid
  ```

JADE Tables

This section describes the **JadeTableElement** subclass changes in this release.

JadeTableColumn Class maxColumnWidth Property (NFS 63804)

The **JadeTableColumn** class now provides the **maxColumnWidth** property, which has an **Integer** value. This property specifies the maximum width (in pixels) for a column when determining the width during the column width auto-size processing.

The default value is zero (**0**), with values in the range zero (**0**) through **32767** pixels permitted. The default value of zero (**0**) means that there is no maximum width and the column will be as wide as required by the content if the column width is auto-sized. If the cell contains a long text string, the column will be as wide as is required to display the entire string.

If the value of the **maxColumnWidth** property is greater than zero (**0**) and column width is determined by the **autoSize** process, the width is restricted to a maximum value of the **maxColumnWidth** property.

The **maxColumnWidth** property is used only during the **autoSize** process and is ignored if the:

- **Table** class **autoSize** property value is not one of **AutoSize_Both**, **AutoSize_BothColumnMinimum**, **AutoSize_Column**, or **AutoSize_ColumnMinimum**.
- Column width has been set by logic (that is, set by the **Table** class **columnWidth** property or the **JadeTableColumn** class **width** property).
- Value of the **JadeTableColumn** class **widthPercent** property is not zero (**0**).

The **maxColumnWidth** property value does not prevent logic from setting a larger column width, nor does it prevent the user from resizing the column width to be larger than the value of the **width** property.

JadeTableSheet Class widthPercentStyle Property (NFS 39629)

The **JadeTableSheet** class now provides the **widthPercentStyle** property, which has an **Integer** value. This property controls how any columns percentages set on the table sheet using the **JadeTableColumn** class **widthPercent** property are interpreted.

The property values can be one of the new **Table** class constants listed in the following table.

Table Class Constant	Integer Value	Description
WidthPercent_Style_ClientWidth	0	The default value, which specifies that if the value of the **JadeTableColumn** class **widthPercent** property is greater than zero (**0**), the width of the column is calculated using the formula (Table.clientWidth * JadeTableColumn.widthPercent) / 100
WidthPercent_Style_NoSetWidths	1	Specifies that if the value of the **JadeTableColumn** class **widthPercent** property is greater than zero (0), the width of the column is calculated using the formula ((Table.clientWidth - <set widths>) * JadeTableColumn.widthPercent) / 100

In this table, the **<set widths>** value is the sum of all column widths that have been specifically set by user logic or by the user resizing the column, which means that if the **widthPercent** property values of the other columns add up to 100 percent, those columns fully use the remaining horizontal space in the table.

If the value of **<set widths>** is greater than value of the **Table** class **clientWidth** property, the width is calculated as if the value of the **widthPercentStyle** property is zero (**0**).

An example of the **widthPercentStyle** property is shown in the following code fragment.

```
table1.accessSheet(table1.sheet).widthPercentStyle := WidthPercent_Style_
NoSetWidths;
```

JadeHTMLClass::buildFormActionOnly Method (PAR 62632)

Because the **JadeHTMLClass** class **buildFormAction** method in earlier releases could not generate correct HTML in all circumstances, the method has now been replaced with the **JadeHTMLClass** class **buildFormActionOnly** method, which has the following signature.

```
buildFormActionOnly(): String;
```

If you call the new **buildFormActionOnly** method, you must change your HTML or JADE code to generate the form tag **method** attribute wherever you were using the superseded **buildFormAction** method with a non-null value in the **meth** parameter (that is, a value of **"get"** or **"post"**).

The return value of **buildFormActionOnly** method is the same as the return value of the superseded **buildFormAction** method when it was called with a parameter value of **null**.

JadeHTTPConnection Class Constants (NFS 62889)

The **JadeHTTPConnection** class provides the following constants, which were not documented in earlier releases.

Name	Type and Value	Description
Header_Protocol	String = "Protocol"	Protocol header
State_NoData	Integer = 6	No data available; not supported
Verb_CONNECT	String = "CONNECT"	The CONNECT operation; not supported

Name	Type and Value	Description
Verb_DELETE	String = "DELETE"	The DELETE operation; not supported
Verb_HEAD	String = "HEAD"	The HEAD operation; not supported
Verb_OPTIONS	String = "OPTIONS"	The OPTIONS operation; not supported
Verb_PUT	String = "PUT"	The PUT operation; not supported
Verb_TRACE	String = "TRACE"	The TRACE operation; not supported

In addition, the existing **State_Failure** class constant indicates a failed HTTP connection state; for example:

- Open failed
- Send request headers failed
- Invalid verb specified for send headers
- WinINet authentication failure
- WinINet set option failure
- WinINet open/connect failure

JadeRecompileAllMethods Application

This section describes the **JadeRecompileAllMethods** application changes in this release.

JadeRecompileAllMethods Errors (PAR 63227)

The **JadeRecompileAllMethods** application, run from **jadclient**, now returns a non-zero exit code 1183 (*Uncompiled or in error methods remain*) if methods could not be compiled or if any methods did not compile without error.

In addition, the output is also now logged to the **JadeRecompileAllMethods.log** file in the default **log** directory. The log file contains details of the methods that were not successfully compiled.

JadeRecompileAllMethods Optional Parameter (PAR 62539)

In earlier releases, the **JadeRecompileAllMethods** non-GUI client application in the **jadclient** executable recompiled only uncompiled methods or methods in error.

To force a recompilation of all successfully compiled methods, the **JadeRecompileAllMethods** application now has the optional **includeAlreadyCompiled** command line argument, which has a Boolean (**true|false** value), as shown in the following example.

```
jadclient path=c:\jade\system ini=c:\jade\system\testjade.ini schema=RootSchema
app=JadeRecompileAllMethods endJade includeAlreadyCompiled=true
```

As the **includeAlreadyCompiled** argument defaults to **false**, specify **includeAlreadyCompiled=true** on the command line to recompile all previously (successfully) recompiled methods.

Locking

This section describes the locking changes in this release.

Background Process Lock Timeout (PAR 62567)

You can now specify a default lock timeout for the background process, in the **BackgroundProcessServerTimeout** parameter in the [JadeClient] and [JadeServer] sections of the JADE initialization file.

The background process lock timeout is specified in milliseconds, with the default value of **30000** (that is, 30 seconds).

Caution You should not change this value unless the background process is having locking problems. Before you increase this value, examine the application to determine which objects are being locked and whether locks are being held for too long. For example, new nodes cannot sign on if the **system.nodes** dictionary is locked by the application. It is better to change the application to minimize the locking of system collections.

Debugging Lock Exceptions (NFS 63339)

JADE now supports the optional recording of the current call stack when a process locks an object. Any process can retrieve this information while the lock is held; for example, you can use it to help find and resolve locking problems during application development, by tracking down where in the code any long-lived lock was obtained.

This information, which is passed to the lock manager and stored in the lock entry, can be retrieved by any process while the lock is held. When a lock is obtained, the saved information includes each method in the current call stack and the call position (source code offset) within each method. You can use this information to produce a call stack summary similar to that shown when you click the **Debug** button on the Unhandled Exception dialog.

Notes The values of local variables are not available, as the code is no longer executing.

This feature is intended for you to use when developing and testing applications. Because of the overhead involved in capturing and saving the extra information, we do not expect that this feature is permanently enabled in a production system.

Automatically enable the debugging of lock exceptions for all client processes on startup, by specifying the **DefaultProcessSaveLockCallStack** parameter with a value of **true** in the [JadeClient] section of the JADE initialization file. To enable the automatic debugging of exceptions for server applications on the database server, specify this parameter and value in the [JadeServer] section of the JADE initialization file. (The default value is **false** on both client and server nodes.)

Dynamically enable and manage the debugging of lock exceptions for a process by calling the methods (documented in the *JADE Encyclopaedia of Classes* (Volume 2)) summarized in the following table.

Class	Method	Description
Object	getLockCallStack	Returns the lock call stack of the process that locked the object.
Process	setSaveLockCallStack	Controls whether lock call stack information is saved for a process. This method returns **true** if the lock call stack was being saved for a process before this method was called; otherwise **false**. (Calling this method overrides the **DefaultProcessSaveLockCallStack** parameter value specified in the JADE initialization file.)
Process	getSaveLockCallStack	When an object is locked, returns **true** if the lock call stack is being saved for a process; otherwise **false**.
Process	addLockCallStackFilter	By default, the lock call stack is saved for all objects that the process locks. This method enables you to define a filter that restricts the saving of this information to instances of specific classes only.
Process	clearLockCallStackFilter	Clears the list of classes for which saving lock call stack information has been enabled for this process.
Process	getLockCallStackFilter	Returns the list of classes for which saving lock call stack information has been enabled for this process.

The JADE Monitor **Users** view now provides the **Enable Save Lock Call Stack** and **Disable Save Lock Call Stack** commands in the popup menu when you right-click on a user and the **Locks** view provides the **Show Lock Call Stack** command in the popup menu when you right-click on a locked option.

Programmatically Changing the Process Lock Timeout (NFS 62581)

You can now programmatically change the default lock timeout period for a process, by calling the **Process** class **setDefaultLockTimeout** method to specify the default lock timeout value for the receiving process. This method has the following signature.

```
setDefaultLockTimeout(timeout: Integer): Integer;
```

The value of the **timeout** parameter can be an explicit number of milliseconds (for example, specify a value of **30000** for 30 seconds), or it can be one of the predefined global constants defined in the RootSchema **LockTimeouts** category; that is, one of:

- LockTimeout_Immediate
- LockTimeout_Infinite
- LockTimeout_Server_Defined

When the **setDefaultLockTimeout** method is called, the parameter value is used for the timeout for all implicit locks and locks obtained by the **Object** class **exclusiveLock**, **sharedLock**, **reserveLock**, and **updateLock** methods.

This method returns the current process lock timeout (before the new value specified in the **timeout** parameter is saved).

By default (that is, if you do *not* call this method), the default lock timeout for all processes is defined by the value of the **ServerTimeout** parameter in the [JadeServer] section of the JADE initialization file.

MemoryAddress Value Adjustment (PAR 62517)

The **MemoryAddress** primitive type now provides the **adjust** method, which has the following signature.

```
adjust(offset: Integer64): MemoryAddress;
```

This method adjusts the value of the **MemoryAddress** of the receiver by an integral amount (for example, when stepping through blocks of memory) and returns a new **MemoryAddress** value.

Method Signature Change (PAR 63453)

The signature of the following methods has changed. The type of the **pNames** or **names** parameter that was **StringArray** in earlier releases is now type **JadeIdentifierArray** (which may cause a warning when upgrading to this release from a JADE 7.0 release).

- Schema::getWebServiceConsumerNames
- JadeAuditAccess::getClassPropertyNames
- JadeAuditAccess::getChangedPropertyNames

Monitoring Web Applications (NFS 52424)

In earlier releases, when JADE Web Application Monitor logging was turned on and the **LogFileName** parameter was specified in the [WebOptions] section of the JADE initialization file, the output includes the content of the Web message, which could include personal and privileged information; for example, user codes and passwords.

You can now specify the new **LogMessageContent** parameter in the [WebOptions] section of the JADE initialization file. Alternatively, you can specify the XML **logmessagecontent** parameter in the Web application configuration file; that is, the following configuration file parameter disables the logging of message content.

```
<logmessagecontent>false</logmessagecontent>
```

The default Boolean value of **true** for the **LogMessageContent** parameter in the [WebOptions] section of the JADE initialization file or the XML **logmessagecontent** parameter in the Web application configuration file applies only if the **DisableLogging** parameter in the [WebOptions] section of the JADE initialization file or the **disable_logging** element in the Web application configuration file is set to **true**, or tracing is turned on when the **Enable Logging** command is displayed in the View menu of the JADE Web Application Monitor window. (The value of this command toggles between **Enable Logging** and **Disable Logging**.)

Note By default, the value of the **DisableLogging** parameter is **false**; that is, logging is enabled.

When the **LogMessageContent** parameter in the [WebOptions] section of the JADE initialization file or the XML **logmessagecontent** parameter in the Web application configuration file is set to **true**, any Web logging includes the **Query String =** ***content*** and **Http String =** ***content*** output, as was displayed in previous releases. When the value of the **LogMessageContent** initialization file parameter or the XML **logmessagecontent** configuration file parameter is **false**, any web logging does *not* include this message content.

When you initiate the Web Application monitor and tracing is on, a message is displayed indicating the status of the **LogMessageContent** parameter, as follows.

```
Message content logging is enabled|disabled
```

The message is also displayed if you turn logging on or off by toggling the **Disable Logging**|**Enable Logging** command in the View menu of the Web Application Monitor window.

Node Object Type Values (PAR 64128)

In earlier releases, the table listing the type of the node object returned by the **Node** class **nodeType** method was incomplete.

The following table lists all node types that can be returned by this method, with the **Node** class now providing the **Type_** class constants listed in the first column.

Node Class Constant	Description
Type_Undefined (0)	Undefined
Type_DatabaseServer (1)	Database server (**jadrap** or **jadserv**)
Type_ApplicationServer (2)	Application server (**jadapp** or **jadappb** in multiuser mode)
Type_ApplicationServerAndDatabaseServer (that is, Type_ApplicationServer + Type_DatabaseServer)	Application server and database server (**jadapp** or **jadappb** in single user mode)
Type_StandardClient (4)	Standard client node (**jade** in multiuser mode; not as a thin client)
Type_StandardClientAndDatabaseServer (that is, Type_StandardClient + Type_DatabaseServer)	Standard client node and database server (**jade** in single user mode)
Type_NonGuiClient (16)	Non-GUI (**jadclient**) node
Type_NonGuiClientAndDatabaseServer (that is, Type_NonGuiClient + Type_DatabaseServer)	Non-GUI (**jadclient**) node and database server
Type_DatabaseAdmin (32)	Database administration (**jdbadmin**) node
Type_DatabaseAdminAndDatabaseServer (that is, Type_DatabaseAdmin + Type_DatabaseServer)	Database administration (**jdbadmin**) node and database server

Non-GUI nodes include user-written executables that use the JADE Object Manger API (C++) and the JADE .NET API (C#).

Notification Data Limits Clarification (PAR 62863)

The documentation in earlier releases did not state that the limit of 48K bytes applied to notifications containing binary or string (**Binary**, **String**, **StringUtf8**) data only when data is sent across the network.

For applications running within the server node, the limit for notifications containing binary or string (**Binary**, **String**, **StringUtf8**) data is 2G bytes. Note, however, that this applies only to single user and server applications. In multiuser applications, persistent notifications are sent via the database server, even if the receiving process is on the same node as the sender.

In notification cause events, exception 1267 (*Notification info object too big*) is raised if the binary or string **userInfo** data exceeds the applicable value.

Object Class autoPartitionIndex Method (PAR 63695)

The **Object** class **autoPartitionIndex** method cannot be used if the database file for that object is encrypted, as the database cannot invoke the **autoPartitionIndex** method using an encrypted buffer. If this occurs, exception 3009 (*File encrypted and partition unspecified*) is raised.

If the file is encrypted, use the **Object** class **setPartitionID** or **setPartitionIndex** method to explicitly set the partition in the created object.

Relational Population Service OID Mapping (NFS 62231)

In earlier releases, the default value in the **OID Mapping Options** combo box on the **Define RPS** sheet of the Relational Population Service wizard was **Map to String**.

As the maximum length of JADE entities increased from 30 to 100 in JADE 7.1, JADE now supports two RPS object identifier (oid) length mapping options: **Map to String (7.0 format)** and **Map to String (7.1 format)**. The 7.0 format size is 16, and the 7.1 format size is 28.

If you want to use the recommended 7.1 format oid string mapping in a new RPS mapping, select the **Map to String (7.1 format)** option.

Notes If you select the **Map to String (7.1 format)** option in an existing RPS mapping, it causes a reorganization and the generation of table alter scripts.

You cannot extract a schema from a system with the **Map to String (7.1 format)** option set and load it into a JADE system that has a different oid mapping. (As the schema load will fail, you must upgrade the system into which you want to load the schema.)

Reorganization when Properties are Renamed (PAR 57289)

In earlier releases, the reorganization process always used property names when comparing the current and latest class definitions. This meant that data was lost if a property was renamed when the class was versioned. While this problem could be avoided by ensuring that all property rename operations were performed on an unversioned class, it was not always possible to ensure the correct order when schema changes were deployed.

The reorganization process has been changed to recognize if the property name has been changed, when determining what schema changes need to be applied.

REST Services

This section describes the REST service changes in this release. (See also "Standalone JSON Functionality (NFS 57762)" and "JadeWebServiceConsumer Class Methods (NFS 63553)", elsewhere in this document.)

File-Related Methods (NFS 64331)

The **JadeRestService** class now provides the following methods, which must be called from an end point method being executed when called from a REST service request.

```
createVirtualDirectoryFile(filename: String;
                           contents: Binary;
                           retain: Boolean): Integer;

deleteVirtualDirectoryFile(filename:         String;
                           deleteIfReadOnly: Boolean): Integer;

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

These method are the same as those in the **JadeWebServiceProvider** class, with the same descriptions and meaning.

JadeRestService::getServerVariable (PAR 63247)

The **JadeRestService** class now provides the **getServerVariable** method, which has the following signature.

```
getServerVariable(var: String): String;
```

This method returns the specified HTTP header information for your REST service request from the Internet Information Server (IIS). This method must be called during the processing of a REST service message; for example, from a re-implementation of the **JadeRestService** class **processRequest** method.

Calling the method when a message is not being processed results in null always being returned. In addition, the method must be called on the same node as the application. If you call the method from a server method and the application is not running on the server, a 31039 error (*Connection invalid invocation*) occurs when trying to access the TCP/IP connection, and error 1242 (*A method executing in another node was aborted*) is reported to the REST service.

As the **var** parameter is IIS-dependent, it is therefore subject to change. Refer to the **ServerVariables** function in your Internet Information Services (IIS) documentation for details. Common server environment variables, documented in the IIS documentation under the **ServerVariables** function, include those listed in the following table.

Variable	Returns…
HTTP_ACCEPT_LANGUAGE	A string describing the language to use for displaying content
HTTP_USER_AGENT	A string describing the browser that sent the request
HTTPS	**ON** if the request came in through a secure channel (SSL) or it returns **OFF** if the request is for a non-secure channel
REMOTE_ADDR	IP address of the remote host making the request
SERVER_NAME	Host name, DNS alias, or IP address of the server as it would appear in self-referencing URLs
SERVER_PORT	Port number to which the request was sent
URL	Base portion of the URL

The method in the following example returns the IP address of the REST service as determined by IIS.

```
processRequest(httpIn: String; queryStr: String; pathIn: String; methodType:
String) updating;
vars
    str : String;
begin
    str := self.getServerVariable("ALL_HTTP");
```



```
        inheritMethod(httpIn, queryStr, pathIn, methodType);
    end;
```

Parsing JSON Text (NFS 63367)

Standard JSON syntax can include the type of the objects to create; for example:

```
/> "__type":"Customer",  .in Microsoft JSON format

/> "$type":"Customer",  .in NewtonSoft (JSONN) format
```

This **"__type"** special type tag must appear as the first entry following the **{** or **[** symbol that begins the object contents description in Microsoft format. In NewtonSoft, the **"$type"** tag can appear after the object reference tag; for example:

```
"id":"12", "$type::"Customer",
```

In NewtonSoft, the type can also include a namespace such as **"MyNameSpace.Customer"**, which JADE will ignore.

If the JSON does not include the type tag, the type of the object is assumed to be the implied type of the expected object; for example, the type of the method parameter to be populated or the type of the property reference.

If the JSON includes the type tag, the type must be the implied type or a subclass of the implied type. If it is not, an exception is generated.

REST Service Process Requests (PAR 62784)

In 7.1 releases prior to 7.1.06, if the **JadeRestService** class **processRequest** method was called by user logic and the call was not part of the JADE REST service web message handling framework, a JADE crash could have resulted; for example, when creating a transient instance of a **JadeRestService** subclass and calling the **processRequest** method. (This occurred because the JADE REST service had not been initialized and the logic assumed that it had.)

To enable manual testing of **JadeRestService** methods, the handling has been changed to allow such a call, as follows.

- The REST service will be initialized if it has not been already, unless the application is already attached to a JADE Web Service Manager. Exception 11126 *(A Rest Service method was called but the service was never initialised)* will result.
- If the **JadeRestService** class **reply** method is called and its processing is not associated with a received web message, exception 11127 *(JadeRestService.reply was called but there is no web message to reply to)* is raised.

  Re-implement the **reply** method on the **JadeRestService** subclass that is being used. When using manual testing processing, to avoid raising exception 11127, the **reply** method should not call **inheritMethod**.

Schema and Class Deletion (NFS 62273)

Documentation in earlier releases did not state that instances of **JadeDynamicObject** and **DynaDictionary** are not deleted if classes referenced by **JadeDynamicObject** or **DynaDictionary** instances are deleted.

In a **JadeDynamicObject**, if the type of a property that has been assigned a value is removed by deleting the class or removing the schema, the value is no longer valid and attempting to use it will raise exception 1046 *(Invalid class number)*.

If the membership type of a **DynaDictionary** is removed by deleting the class or removing the schema, any dynamic dictionaries that have been populated with that membership class are no longer valid and attempting to use it will raise exception 1046.

Setting Menu Accelerators at Run Time (NFS 64000)

The **MenuItem** class now provides the **setAccelerator** method, which has the following signature.

```
setAccelerator(key: Character; flags: Integer);
```

This method enables you to set the accelerator displayed for a menu item at run time. (In earlier releases, you could define accelerators only in the JADE Painter during development.)

The specified accelerator is added to the menu item displayed on the owning form or it replaces an existing accelerator. The accelerator is ignored if the menu item is a popup menu or it is the top-level menu of the form, as these do not accept accelerators.

The method generates an exception if the parameters are invalid.

The value of the **key** parameter must be one of **"0"** through **"9"**, **"A"** through **"Z"**, **J_key_F1** though **J_key_F12**, **J_key_Delete**, **J_key_Insert**, **J_key_Back**, **J_key_UpArrow**, or **J_key_DownArrow**. (The **J_key_j** values are global constants in the **KeyCharacterCodes** category.)

The value of the **flags** parameter can be zero (**0**) if there is no shortcut flag or it can be a combination of the following **MenuItem** class constants.

Constant	Bit Value	Description
ShortCutFlag_Alt	#10	The Alt key must also be pressed
ShortCutFlag_Ctrl	#8	The Ctrl key must also be pressed
ShortCutFlag_Shift	#4	The Shift key must also be pressed

The code fragment in the following example displays Shift+Ctrl+Alt+Delete as the menu accelerator.

```
menu1.setAccelerator(J_key_Delete.Character, MenuItem.ShortCutFlag_Alt +
                     MenuItem.ShortCutFlag_Ctrl + MenuItem.ShortCutFlag_Shift);
```

Skin Maintenance at Run Time

The Jade Skin Maintenance dialog has been enhanced, as follows. (For details about using skins in runtime applications, see "**Defining and Maintaining JADE Skins at Run Time**", in Chapter 2 of the *JADE Runtime Application Guide*.)

- The form has been enlarged and the layout of each sheet has been spaced out more.
- The **Categories** sheet has a new list box that is filled with the list of skin entities that use the currently selected category.
- The **Users of a Skin Entity** sheet has a:
  - New list box that shows all of the skin entity children referenced by the currently selected skin.
  - **Show only Unused Entities** check box, which when unchecked, includes all defined skin entities in the entity list.

    When checked, the **Skin Entity List** table contains only skin entities that are not referenced by any other skin entity. In addition, the **All Entity Children** table contains only children that are referenced only by the selected skin entity.

    Note The unused entities list also includes application skins, as these have no skin entity references.
  - **Delete** button at the lower left. This button is enabled only if the selected skin entity is not referenced by any other skin element.

Clicking the enabled button shows a confirmation message box. If you click the **Yes** button, the selected skin entity is deleted (but no referenced children).

- **Delete with Children** button on the lower right. This button is enabled only if the selected skin entity is not referenced by any other skin element.

  Clicking the enabled button shows a confirmation message box. If you click the **Yes** button, the selected skin entity is deleted together with all child skin entities that are were referenced only by the selected skin entity or its children.

Standalone JSON Functionality (NFS 57762)

JADE now provides the **JadeJson** class, which is a transient-only **Object** subclass that provides standalone JSON functionality that is independent of the Representational State Transfer (REST) Application Programming Interface (API).

The **JadeJson** class enables you to create, load, unload, and parse JSON in the same way you can with XML.

The methods defined in the **JadeJson** class are summarized in the following table.

Method	Description
generateJson	Generates JSON from a primitive type variable or an object
generateJsonFile	Generates JSON from a primitive type variable or an object and writes the output to a file
parse	Parses JSON text to create and populate an object and all referenced objects
parseFile	Reads and parses JSON text from a file to create and populate an object and all referenced objects
parsePrimitive	Parses JSON text for a primitive type and returns the primitive type value
parsePrimitiveFile	Parses JSON text for a primitive type from a file and returns the primitive type value

The constants provided by the **JadeJson** class are listed in the following table.

Class Constant	Integer Value	Description
Format_Json_Microsoft	0	The JSON format as is expected by the Microsoft **DataContractJsonSerializer** class. This format type does not support circular references or multiple references to the same object in the returned data (an exception is generated if the situation is detected).
Format_Json_Newton	2	The JSON format is as expected by the NewtonSoft **Json** class software. This format is different from Microsoft in the structure, tags, and the format of some primitive types. The output includes ids for each object and references to already included objects, and therefore supports circular and multiple references to the same object in the returned data.

In general, the standalone JSON functionality uses the same error codes as REST services JSON handling. The following errors are new in this release.

- 11151 *(Cannot access a JadeJson object created by a different node*), which is caused if you attempt to reference a **JadeJson** object created on a different node. If this error is raised, fix your logic.
- 11152 (*The Json cannot be generated because the same object is referenced twice*), which is caused if you attempt to generate the JSON for an object tree where the same object is referenced twice and the format is **Format_Json_Microsoft**. If this error is raised, use the **Format_Json_Newton** (NewtonSoft) formatting or change the object structure.

- 11153 (*The type found in the Json is unknown*), which is caused when a type specified in the JSON is not a known JADE class. If this error is raised, fix the task generating the JSON so that it specifies the correct JADE class.
- 11154 (*The type of the object to decode is invalid*), which is caused when the type of the primitive data to be parsed is invalid. If this error is raised, fix your logic.

For details about using the standalone JSON functionality provided by the **JadeJson** class, see the **JadeJson** class in volume 1 of the *JADE Encyclopaedia of Classes*. See also "Parsing JSON Text (NFS 63367)", earlier in this document.

String Primitive Type asUuid Method (NFS 62803)

The **String** primitive type now provides the **asUuid** method, which has the following signature.

```
asUuid(): Binary;
```

The **asUuid** method returns a binary by formatting the string as a Universally Unique Identifier (UUID).

If the string is not formatted as a valid UUID representation (that is, as returned by the **Binary** primitive type **uuidAsString** method), exception 1407 *(Invalid argument passed to method)* is raised.

The code fragment in the following example shows the use of the **asUuid** method.

```
vars
    str : String;
    bin : Binary;
begin
    str := "4dfc912a-b466-01d0-1027-000085823b00";
    bin := str.asUuid();
```

Synchronized Database Service (PAR 62798, 57217)

In earlier releases, the *JADE Initialization File Reference* did not state that the [SyncDbService] section could contain the following parameters. In addition, it was not documented that the maximum value of the **JournalXferBlocksize** parameter in the [SyncDbService] section has increased from **512**K to **1**G.

JournalReadBuffers

Value Type Integer

Default 4

Purpose

The **JournalReadBuffers** parameter specifies the number of buffers to use when reading a journal file on disk.

The minimum value for this parameter is **2** and the maximum value is **100**.

Parameter is read when …

The SDS service is next initialized.

Applicable to database role…

Primary (applies to journal transfer).

Secondary (applies to journal replay).

JournalReplayBlocksize

Value Type Integer *prefix multiplier*

Default 128K

Purpose

The **JournalReplayBlocksize** parameter specifies the size in bytes of each read buffer used when replaying a journal file.

The minimum value for this parameter is **4**K and the maximum value is **1**G.

Parameter is read when …

The SDS secondary service is next initialized.

Applicable to database role…

Secondary.

Thin Clients

This section describes the JADE thin client changes in this release.

Downloading Additional Thin Client Binaries (PAR 63174)

When files and directories are downloaded to the presentation client from the application server, the directory structure under the application server download directory (for example, **...download/jade.exe**) for the architecture of the presentation client is duplicated in the JADE installation directory on the presentation client. The files in those directories on the application server are downloaded and copied to the equivalent directory on the presentation client. (The directories are created if they are not present.)

The exception to this, however, is the directory that contains the **jade.exe** executable on the presentation client and the application server download directory that contains the **jade.exe** executable, as those directories are assumed to be the equivalent **bin** directories in both environments. The result is that all of the files in the download **bin** directory of the application server are copied to the presentation client **bin** directory, even if the names are different. In addition all subdirectories of the application server **bin** directory are copied as subdirectories of the presentation client **bin** directory. For example, the:

- Application server has:

  ```
  ......\download\bin\ (which contains jade.exe)

  ......\download\bin\sub-bin\
  ```

- Presentation client has:

  ```
  <JADE-install-directory>\mybin\ (which contains jade.exe)
  ```

The result is that the files in **bin** are copied to **mybin** and a **sub-bin** directory is created as a subdirectory of **mybin**.

Secure Sockets Library (PAR 63548, NFS 63550)

JADE supports the 1.0.2g-level OpenSSL libraries, which removes support for the insecure **SSLv2** method. In addition, JADE has removed support for the insecure **SSLv3** method.

From JADE 7.1.07, JADE accepts connections using only TLSv1 or above. The TLSv1.2 method is now the JADE default value. This has the following effects on existing JADE systems.

- If a value of **SSLv2**, **SSLv23**, or **SSLv3** in the **SSLMethodName** parameter is present in the [JadeAppServer] or [JadeThinClient] section of an existing JADE initialization file, it is overwritten with **<default>**, a message is

written to the **jommsg** log file, and JADE attempts to make an SSL connection using the new **TLSv1.2** default method. This connection can fail for the following reasons.

- Existing X509 certificates can be rejected by the OpenSSL libraries if they are incompatible with the upgraded requirements of **TLSv1** or higher.

  Note As the previously distributed versions of the **server.pem** and **client.pem** OpenSSL insecure example certificates are subject to this condition, a new version of the example certificates is provided with this release.

- Existing connections can be refused if the explicit list of ciphers defined in the JADE initialization file are incompatible with the upgraded requirements of **TLSv1** and higher.

  Tip Unless you have special requirements, leave the value of the **SSLCipherNames** parameter blank, to use the default, compatible list of ciphers provided by the OpenSSL libraries.

To enhance SSL security, the default values of the **SSLMethodName** and **SSLCipherNames** parameters in the [JadeAppServer] and [JadeThinClient] sections of the JADE initialization file are now:

- SSLMethodName=TLSv1.2
- SSLCipherNames=Not specified (compatible ciphers are available from the **OpenSSL** online documentation or **openssl.exe**)

See also "OpenSSL Library Implementation (PAR 63548)" under "JADE 2016 Changes that May Affect Your Existing Systems", earlier in this document.

Time Millisecond Value (PAR 63572)

The millisecond value of a time expressed as a string represents a fraction of a second, but in earlier releases, JADE assumed it was the number of milliseconds. Any value that was greater than 999 was rejected as being invalid. Any value where the value was one or two digits was wrongly converted; for example, **'.1'** represents 1/10 of a second (100 milliseconds), where previously JADE would have treated it as 1 millisecond.

JADE now handles millisecond values greater than three digits.

Unhandled Exceptions (NFS 63168)

The error object reported by the default exception handler now includes the type name before the oid if the class number is valid; for example:

```
...
Error item: setFontProperties
Error object: TextBox/509.21 (transient)
Caused By:
   Receiver: MainForm/1004290.1 (transient)
   Method: MainForm::setupClipText(1037)  --           tb.setFontProperties
   (tblClipBoard.fontName, tblClipBoard.fontSize, tblClipBoard.fontBold);
Reported By:
   Receiver: TextBox/509.21 (transient)
   Method: Control::setFontProperties  -- 'JadeControlSetFont' in 'jadpmap'
...
```

If there is no class in the current system that has the specified class number, only the oid is displayed, as was the case in earlier releases.

Upgrade Validation

This section describes the upgrade validation changes in this release.

Method Recompilation Failure (PAR 63810)

In a major JADE release, some changes to the JADE model or interpreter engine require methods that make use of the changed feature to be rebuilt, by recompiling the methods. The upgrade process fails if methods cannot be recompiled.

In earlier releases, the failure of any method to compile failed the upgrade validation and the process exited with error 8701 (*General Upgrade error code*). The **jadloadb** or **jadclient** upgrade process now exits with error 8711 (*Uncompiled or in error methods were detected in this database*) if recompiling one or methods fails, making it clear that you must reload some methods and restart the validation.

Upgrade Validation in a Source-Stripped Database (PAR 63458)

In earlier releases, the upgrade validation failed if the source was not present or was encrypted for a method that needed to be recompiled.

When upgrading a database that has method source stripped, the methods that need to be recompiled are now marked in error and the upgrade process is allowed to complete and error 8703 (*Method could not be recompiled - no source present*) is raised, indicating that method source needs to be loaded and the methods recompiled. The list of methods that could not be compiled is output to the **jadeupgrade.log** file. When the methods marked as requiring source recompilation have been successfully recompiled, the upgrade of the system does not need to be re-validated after any subsequent schema loads if error 8703 is raised.

The upgrade process now recompiles encrypted methods and the upgrade does not fail if the re-compilation succeeds.

Web Services

This section describes the Web services changes in this release. (See also "REST Services" and "Monitoring Web Applications (NFS 52424)", elsewhere in this document.)

Defining Web Services in an Application (NFS 63057)

The Define Application dialog has been changed, as follows.

- The **Web Services** sheet of the **Web Options Folder** sheet now includes the following list boxes.
  - Web Service Exposures Available
  - Web Service Exposures To Use

  The right arrow (**>**) and left arrow (**<**) buttons between the two list boxes enable you to add a Web service exposure to or remove it from the list of exposures to use for the application. Alternatively, you can double-click on an exposure in one list to move it to the other list. An exposure selected for inclusion in the application is displayed in blue text in the **Web Service Exposures To Use** list box and disabled in the **Web Service Exposures Available** list box.
- You can now resize the Define Application dialog. When resized, the Web service exposure list boxes are resized and repositioned to make use of the extra space.

When changing an existing application, the list of exposures to use for the application is automatically loaded into the **Web Service Exposures Available** list box on the right of the dialog and the matching entries disabled in the **Web Service Exposures To Use** list box at the left.

Note The previous mechanism of selecting one or more Web services from the **Web Service Exposures** list box on the **Web Services** sheet in earlier releases no longer applies.

Generating a Web Consumer Test Case (NFS 64085, NFS 64157)

In earlier releases:

- The Classes menu in the Class Browser provided the **Generate Test Case** command, which was disabled if the selected class was not a **JadeWebServiceConsumer** subclass.

  As this functionality generates test cases only for Web service imported classes, this has been renamed the **Generate Web Consumer Test Case** command and it is now displayed only if the selected class is a subclass of the **JadeWebServiceConsumer** class.

- After generating Web service consumer test cases, attempting to repeat the generate process to add additional test case methods failed, with a message box stating that the test case class already existed.

  The Generate Web Consumer test case functionality has now been enhanced, as follows.

  - The form has been enlarged and can now be resized.
  - If the selected schema is versioned and you are working in the latest version, the existing **JadeTestCase** subclass is versioned if it is unversioned.
  - The generate function now handles the case where the generation process is being performed for the same **JadeWebServiceConsumer** subclass into an existing **JadeTestCase** class, as follows.
    - It creates the required Web service consumer class property reference on the class, if it does not exist.
    - It creates any methods selected in the displayed table that do not exist in the specified **JadeTestCase** subclass.
    - It creates the **create** method, if it does not exist.
    - If the **create** method already exists, you are warned if it does not contain the **create** ***property-name web-service-consumer-class-name*****;** statement.

      You must manually add that statement to the **create** method.
    - It ignores selected methods when a method of that name already exists in the **JadeTestCase** subclass.
    - If selected methods are ignored because a method of that name already exists, a warning is displayed in a message box on completion of the generate process.

JadeWebServiceConsumer Class Methods (NFS 63553)

You can now specify the verb when invoking the **JadeHTTPConnection** class. When the **JadeWebServiceConsumer** class **invoke** and **invokeAsync** methods are called, the message is sent via the associated **JadeHTTPConnection** using the **POST** verb.

To allow the HTTP verb to be specified, the **JadeWebServiceConsumer** class now provides the following methods.

- invokeWithVerb(inputMessage: String; verbIn: String): String;
- invokeAsyncWithVerb(inputMessage String; verbIn: String): JadeMethodContext;

These method do the same as the **JadeWebServiceConsumer** class **invoke** and **invokeAsync** methods, respectively, except that the verb specified in the **verbIn** parameter is used (for example, **"GET"** or **"PUT"**) instead of **"POST"**. Calling the **invoke** or **invokeAsync** method is the same as calling the **invokeWithVerb** or **invokeAsyncWithVerb** **invokeAsyncWithVerb** method with **"POST"** specified in the **verb** parameter.

You can use the **invokeWithVerb** and **invokeAsyncWithVerb** methods, for example, to allow users to access a REST service using the HTTP **GET** verb.

Web Service Exposure Error Message (PAR 63094)

In earlier releases, if a Web service encountered a property whose instance was a subclass of the defined class type and that subclass was not included in the exposure, an assertion failure occurred.

This has been changed so that a new exception (11058) is now raised. If this exception is raised, update the exposure to include the subclass.

Web Service WSDL (NFS 64042)

In earlier releases, fields except for string were marked **minOccurs="0"** in the Web Services Description Language (WSDL) generated by JADE for a Web service provider, as shown in the following WSDL file snippet.

```
<xsd:sequence>
    <xsd:element minOccurs="0" name="offer" type="tns:decimal_12_2" />
    <xsd:element minOccurs="0" name="timeStamp" type="xsd:dateTime" />
</xsd:sequence>
```

This caused problems if a software version at a remote site strongly validated against the XAML Schema Definition (XSD). Because string fields did not have a **minOccurs="0"** attribute, the validation failed when the attribute was missing from a string field.

JADE now also marks string elements with minOccurs="0".

Window Collections (NFS 40063)

In earlier releases, if you wrote code to iterate all of the children of a specific container control, you had to iterate all of the controls on the form and check if the parent control was a child of the specific control.

You can now write code that iterates the **Window** class **Children** collection recursively, to achieve the required result for all child controls of a specific control. JADE now provides the following array collection properties, which are accessed at run time only.

- **Window** class **controlChildren** property, which is a reference to an array of all the immediate form or control children of the window; that is, the window is the direct parent of each control in the collection.

  Note The list is in no particular order.

  The list is changed if the z-order of a control is changed by logic and if the parent of a control is changed.

- **Window** class **allControlChildren** property, which is a reference to an array of all controls that are contained in that window (**Form** or **Control**), including children of children.

  Note The list is in no particular order except that children come after parents.

  The list is changed if the z-order of a control is changed by logic and if the parent of a control is changed.

- **Form** class **allMenuItems** property, which is a reference to an array of all menu items on the form. The value of this property is the equivalent of using the **Form** class **menuItems** method. The collection is ordered according to the defined menu item list.

- **Form** class **topLevelMenuItems** property, which is a reference to an array of all of the top-level menu items on the form (that is, those that would appear on the form menu bar). The collection is ordered according to the defined menu item list.

- **MenuItem** class **children** property, which is a reference to an array of all of the immediate children of the **MenuItem** instance; that is, the **MenuItem** instance is the direct parent of the menu items in the collection. The collection is ordered according to the defined menu item list.

- **MenuItem** class **allChildren** property, which is a reference to an array of all menu item children contained in the menu item, including children of children. The collection is ordered according to the defined menu item list.

The arrays include controls and menu items that are invisible.

The following code fragments are examples of the use of Window array properties.


```
foreach menu in mnuOptions.children do
    if menu.checked then
        .... // do some processing here
    endif;
endforeach;

foreach control in frameLeft.allChildren do
    if control.isKindOf(Label) then
        control.fontBold := true;
    endif;
endforeach;
```

