Upgrading to the JADE 2018 Release

VERSION 2018.0.01

Contents

Upgrading to the JADE 2018 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 2018 is built using Microsoft Visual Studio 2017, which requires the installation of the Microsoft Visual C++ 2017 Redistributables.

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.

MenuItem Class setAccelerator Method Deprecation

The **MenuItem** class **setAccelerator** method has been deprecated in this release, and will be deleted in a future JADE release.

Note Use the **MenuItem** class **setShortCutKey** method instead of the deprecated **setAccelerator**method, as the new method more accurately defines the method behavior.

For details about changing accelerator keys, see "Handling Shortcut Keys" under "Changes and New Features in JADE Release 2018".

Highlights in this Release

The highlights in JADE release 2018, which help you to deliver high-performance, interoperable applications on Windows for both 32-bit and 64-bit platforms, are as follows. You can now:

- Use the built-in Git client source control to access a basic set of Git commands. You can use these to store and retrieve files already extracted from the JADE development environment to and from a Git repository.

  The JADE Git client functionality is supported on both presentation and standard clients. It is fully compatible with other Git clients that you can use for more-specialized operations.

  Note The JADE Git client does not extract or load files directly into or from JADE itself. You must use existing JADE development environment functionality to perform these operations.

  For details, see "Source Control", later in this document.

- Use constructors with parameters, to create an object in a single step. For details, see "Constructors with Parameters (NFS 64738, JEDI 20)", later in this document.

- Extract and load form and data definition files in the XML Device Data Exchange (DDX) format instead of the default existing Device-Dependent Bitmap (DDB) format. For details, see "Extracting and Loading Definition Files in DDX Format (NFS 66195)", later in this document.

- Customize the layout of browser forms. For details, see "Customizing the Layout of Hierarchy Browser Forms (NFS 66119)", later in this document.

- Maintain text box hint text and color and a number of skins enhancements. For details, see "Text Box Hint Text and Color (NFS 65829, JEDI 117)" and "Skins", respectively, later in this document.

- Record and replay GUI actions in JADE applications by capturing the execution of GUI event methods, and generating code to replay those actions. For details, see "Automated Test Code Generator (ATCG) Reference", elsewhere in this document.

- Analyze coding errors using AutoComplete. For details, see "Enhancements and Method Error Analysis (NFS 65977)", later in this document.

- Create a backup of a secondary that when recovered, stops tracking and is in a commit-consistent (coherent) state. For details, see "SDS Secondary Commit-Consistent Backup (NFS 66069, JEDI 160)", later in this document.

- Load schemas in an application server in single-user mode. For details, see "Loading Schemas in a Single-user Application Server (NFS 64795, 65133)" and "JadeSchemaLoader Application Security Considerations", later in this document.

- Identify oids added to and removed from a collection block update record. For details, see "Audit Access (NFS 64273)", later in this document.

- Get and send the method cache statistics of the executing process. For details, see "Getting and Sending Method Cache Statistics (NFS 64890)", later in this document.

- Return a new string that has had the specified replacement made. For details, see "String Replacement Methods (NFS 65528, JEDI 2)", later in this document.

- Use the Relational Population Service (RPS) with SQL Server 2016. For details, see "SQL Server 2016 Support (PAR 66028)", later in this document.

- Use the WebSocket protocol for two-way communication over a single TCP connection. For details, see "WebSocket Protocol", 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 **2018** list item).

3. Click the **Search** button.

Upgrading to JADE 2018

This section covers the following topics.

- Upgrading to JADE 2018 from JADE 2016
    - Running Two Releases of JADE on the Same Workstation
- JADE Thin Client Upgrade
- Upgrading an SDS Native or RPS Secondary System
- Upgrade Validation

Caution Before you upgrade to JADE 2018, refer to "JADE 2018 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 2018 from JADE 2016

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. They must be downloaded from the appropriate link (for example, **JADE-Erewhon** or **JADE-ATCG**) at https://github.com/jadesoftwarenz.

The JADE Setup program enables you to upgrade your binary and database files to JADE 2018 from JADE 2016, by performing the following actions.

1. On the JADE 2016 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 2016 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. If you defined your own **String** primitive type **replace__** and **replaceFrom__** methods in JADE 2016.0.02 or later, you must rename them *before* you upgrade to JADE 2018, as JADE's new **replace__** and **replaceFrom__** methods will conflict with any existing **String** method named **replace__** or **replaceFrom__**.

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

5. Run the JADE 2018 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.

6. 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.

7. 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.

8. Use the JADE Database utility to take a full backup of your JADE 2018 database.

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 2018, 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 2018. 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.

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.

JADE 2018 Changes that May Affect Your Existing Systems

This section describes only the changes in the JADE 2018 release that may affect your existing systems. Some changes may result in compile errors during the load process, or cause your JADE release 2018 systems to behave differently.

Accessing .NET Events Defined in Superclasses (NFS 63536)

The JADE .NET import assembly has been enhanced as follows.

- The import process can now optionally import the super-classes referred to by classes in the assembly when the super-classes are in other assemblies.
- Re-importing an assembly now deletes entities that are no longer defined in the assembly.
- If a class is re-parented in the assembly, the re-import process will move the existing class to the new parent.

The first sheet of the .NET Import Wizard sheet that enables you to select the .NET assembly you want to import into JADE now provides the **Import Super Classes in Other Assemblies** check box, which is checked by default. When this check box is unchecked, the functionality of earlier JADE releases applies; that is, the import process does not include referenced superclasses that are in other assemblies nor events that reference classes that are not in the assembly.

When the check box is checked (that is, **true**), the import process includes superclasses of classes in the assembly when the superclasses are in other assemblies, so that classes, properties, methods, and events defined for the superclasses are included. It also means that all events that refer to those superclass classes are included.

When you change the check box status and re-import the assembly, the schema is versioned if it is not already versioned and classes are moved so that they are parented according to the resulting imported hierarchy structure.

When including superclasses in other assemblies, note the following.

- Moving a class to a new parent is possible only under strict rules.

  If the moving of a class fails, the **jommsg log** will contain a description of the reason for the failure. To achieve the move, it may be necessary to change the existing system before the move can be performed; for example, if the method definition of the manually added method clashes with a method in the new parent hierarchy.

- When an assembly is re-imported and the previous import used a different import philosophy of superclass classes (that is, JADE 2016 effectively used a check box value of **false**), you may need to change existing JADE logic.

  A change in the import style means that method signatures can be different for parameters and return types. The method may be renamed and existing application logic may not compile because the method has been removed (and re-added under another name), or parameters or return types are different.

- If two different imported assemblies refer to the same superclasses from another assembly, each import will have its own set of differently named superclasses.

  They are not shared between the two assembly hierarchies; the shared structure is not possible to achieve because the root class for the assembly holds global constants for the assembly.

- When importing classes from the **System:Windows:Forms** assembly, the classes are split into two separate parts in the hierarchy: **JadeDotNetType** and **JadeDotNetVisualComponent**, according to whether or not the class is a control.

The superclasses of **System.Windows.Forms.Control** (**System::MarshalByRefObject** and **System.Windows.Forms.Component**) are not included in the JADE control hierarchy because the classes are not controls. However, the properties and methods for these classes are also added to the **Control** class, so they *are* available.

setAccelerator Method Replaced

As stated in "Menu Class setAccelerator Method Deprecated" under "Deimplementations and Deprecations", the **MenuItem** Class **setAccelerator** method has now been replaced by the **setShortCutKey** method.

String Replacement Functionality Updated (NFS 65528, JEDI 2)

The documentation of the string replacement functionality, implemented in JADE 2016.0.02 (Service Pack 1), has been updated.

For details, see "String Replacement Methods (NFS 65528, JEDI 2)", elsewhere in this document.

Changes and New Features in JADE Release 2018

This section summarizes the product and documentation changes and new features in JADE release 2018.01. For details about the changes in release 2018 that may affect your existing systems, see "JADE 2018 Changes that May Affect Your Existing Systems", earlier in this document.

Application Handling from jadclient (NFS 65845)

The **jadclient** executable now recognizes **startAppParameters** and **endAppParameters** arguments on the command line. These command line options were added to the **jade** executable in an earlier release, to enable the collection of command line arguments into a **HugeStringArray** that is passed to the **initialize** method of the application being run.

The first of the **startAppParameters** or **endJade** argument that is encountered dictates the handling of the application; that is, if:

- **startAppParameters** is encountered first, all subsequent arguments up to the **endAppParameters** argument or the end of the command line if the **endAppParameters** argument is not present are collected into a **HugeStringArray** and passed to the **initialize** method of the application.

  If the **startAppParameters** argument is present, any **endJade** argument is not used for the collection of your own command line arguments other than to terminate the collection of command line arguments after the **endAppParameters** argument.

- **endJade** is encountered first, all arguments following the **endJade** argument are collected into a **HugeStringArray** and passed to the **initialize** method of the application.

For details, see "Running a JADE Non-GUI Client Application with Parameters", in Chapter 1 of the *JADE Runtime Application Guide*.

Audit Access (NFS 64273)

The **JadeAuditAccess** class now enables you to identify oids added to and removed from a collection block update record.

As with Binary Large Objects (blobs) and String Large Objects (slobs), collection block information is made available only when access mode is set to **Jaa_AccessMode_Long** (by calling the **setAccessMode** method).

Collection block information can be retrieved for collections of **Set**, **ObjectArray**, and **MemberKeyDictionary** classes and their subclasses.

The new **Jaa_Object_CollectionBlock** (5) class constant now enables the **getNextRecord** and **getNextRecordUTC** methods to return the collection block object type. When the object type is a collection block, the **oid** parameter of these methods contains the oid of the collection.

You can now also call the following new **JadeAuditAccess** class methods.

- **getCollectionBlockOid** method to return the block oid as a string
- **getCollectionBlockOids** method to have the **addedOids** and **removedOids ObjectArray** parameters populated with the string representations of the oids added and removed in this edition of the collection block
- **descriptionClassIsSubclass** method, which returns **true** if the class number specified in the **classNumber** parameter is a subclass of the class number specified in the **superclassNumber** parameter in the loaded description

When the current oid is a **MemberKeyDictionary** or subclass (that is, **jaa.descriptionClassIsSubclass (recordClassNumber, MemberKeyDictionary.number)** is **true**), you can call the new **getCollectionBlockKeys** method, which populates the passed array with dynamic objects describing the additions, removals, and changes in this edition of the **MemberKeyDictionary** collection block. The attributes on each dynamic object are the entry oid as a String value, the byte length of the keys as an Integer value, and the before and after keys as Binary values.

Entries added to the collection block will have empty before keys, entries removed from the collection block will have empty after keys, and entries with changed keys have before and after keys.

For details, see the **JadeAuditAccess** class in the *Encyclopaedia of Classes (Volume 1)*, and the explanations and examples in the *Audit Access White Paper*.

Automated Test Code Generator (ATCG) Reference

The JADE product information library now contains the *Automated Test Code Generator (ATCG) Reference*, which documents using the Automated Test Code Generator to record and replay GUI actions in JADE applications by capturing the execution of GUI event methods, and generating code to replay those actions.

Binary Type copyImage Method (NFS 65926)

The **Binary** primitive type now provides the **copyImage** method, which has the following signature.

```
copyImage(left:   Integer;
          top:    Integer;
          width:  Integer;
          height: Integer): Binary;
```

This method enables you to create a new image (a rectangle subset) from the specified part of an existing binary image of the receiver. The created image has the same type as the original image from which it was created (that is, a **.bmp**, **.tiff**, **.gif**, **.png**, or **.jpeg** image).

In addition, the created image has the same bit and color depth as the original image (for example, 24-bit, 8-bit, and so on). If the image is 32-bit, the alpha channel (transparency) information is preserved for each pixel description that is copied.

The **copyImage** method generates an exception if the:

- Binary is not a valid **.bmp**, **.tiff**, **.gif**, **.png**, or **.jpeg** image
- Method is called from a server method (which requires **jade.exe** to perform the image copy)
- Application is not a GUI application
- Rectangle specified is not a subset of the total image rectangle

C# Exposure Generation and Extraction Automation (NFS 63711, 65391)

In earlier releases, you could generate (build) and extract a C# exposure using the non-GUI client executable (**jadclient.exe**), but without all of the functionality provided by the JADE development environment. This was available using the following command and parameters.

```
jadclient schema=JadeSchema
          app=CSharpGenerator
          path=database-path
          ini=JADE-initialization-file-path-and name
          startAppParameters
          name-of-existing-C#-exposure
          absolute-path-of-output-directory
```

You can now use the non-GUI client **jadclient** executable to build and extract a C# exposure with the options provided by the C# Exposure wizard; for example, the generation of the **.proj** file. The parameters following **startAppParameters** must be specified in the following order.

```
jadclient schema=JadeSchema
          app=CSharpGenerator
          path=database-path
          ini=JADE-initialization-file-path-and name
```

```
 startAppParameters
 schemaName=top-most-schema-in-which-classes-defined
 exposureName=existing-exposure-name
 outputFolderName=absolute-path-and-name-of-output-directory
[shouldGenerateProjFile=true|false]
[shouldGenerateAppConfigFile=true|false
 dataSourceConfigParam=absolute-path-of-configuration-file
 configFileConfigParam=name-of-configuration-initialization-file
 singleUserConfigParam=true|false
 schemaConfigParam=name-of-schema-configuration-file
[applicationConfigParam=name-of-application-configuration-file]]
```

The following is an example of the **jadclient** command line that generates and extracts a C# exposure.

```
jadclient schema=JadeSchema app=CSharpGenerator path=c:\jade\system
ini=c:\jade\system\jade.ini startAppParameters
schemaName=ErewhonInvestmentsModelSchema exposureName=DocCsharpExample
outputFolderName=c:\jade\csharp shouldGenerateProjFile=true
shouldGenerateAppConfigFile=true dataSourceConfigParam=c:\jade\system
configFileConfigParam=c:\jade\system\jade.ini singleUserConfigParam=false
schemaConfigParam=DocCsharpDemoSchema applicationConfigParam=CsharpConnectionApp
```

Constructors with Parameters (NFS 64738, JEDI 20)

JADE now supports a new extended **create** instruction and allows parameters to be declared on the **create** method, which enables you to create and initialize an object in a single step.

Use of the existing **create** instruction and **create** method continues to be supported, with the exception that the old syntax cannot be used if the class of the object being created has a constructor that has parameters.

This new syntax is allowed for any class, with or without an explicit constructor with or without parameters.

Note As JADE does not support method overloading, a class can still have one explicit constructor only.

For more details, see "create Instruction", in Chapter 1 of the *JADE Developer's Reference*, and "Using Constructors and Destructors", in Chapter 4 of the *JADE Development Environment User's Guide*.

See also the following subsections.

create Instruction

A new syntax for object instantiation using the extended **create** instruction has been implemented. The object reference is retrieved using an assignment instruction, as follows.

```
reference := create class-name(parameters) lifetime;
```

The following code fragment is an example of the extended **create** instruction with parameters.

```
vars
    c : C;
begin
    c := create C(42) transient;
...                 // do some processing here
```

The new **create** instruction syntax is allowed for any class, with or without an explicit constructor and with or without parameters.

For more details, see "create Instruction", in Chapter 1 of the *JADE Developer's Reference*.

create Method

Parameters can now be declared on the **create** method.

Superclass constructors with parameters are called explicitly from the constructor of any immediate subclasses. The syntax of the **create** method with parameters is as follows.

```
create(optional-parameters) [::super(required-parameters)] updating;
```

The optional **::super** value indicates that the superclass constructor is invoked with the specified parameters. For this syntax to be used, the **create** method in the superclass must have parameters. If the superclass constructor has parameters, the compiler verifies that the call to the constructor is present and that the number of parameters is correct and that the types are compatible with the definition of the superclass **create** method If the superclass constructor does not have parameters, an explicit call is not allowed..

Notes Changing the signature of a **create** method results in any methods creating an instance of the class being marked for recompile.

Parameters must be usage **constant** or usage **input**. Usage **io** and usage **output** parameters are not supported.

If a class with a constructor with parameters is exported from a package, the **create** method must also be exported.

JADE does not allow parameters to be declared on the **create** method of classes that derive from any RootSchema class other than **Object**. This restriction is required because when JADE instantiates instances of these classes, the parameters values are not always known. For example, **Form** and **Control** instances are created when a form is displayed, and exclusive collection instances are instantiated when the collection is populated.

For more details, see "Using Constructors and Destructors", in Chapter 4 of the *JADE Development Environment User's Guide*.

Constructor Method Example

The hierarchy in the examples of the new extended **create** method syntax is as follows.

```
A - create() // explicit
  \
    B - create(string: String) // explicit
      \
        C - create() // explicit
```

The following, based on the above hierarchy, are examples of the new method syntax.

```
A - create() updating;
B - create(string: String) updating;
C - create() ::super("my name") updating;
```

When an instance of **B** is created, **A::create()** is called then **B::create()** is called with the parameters provided in the **create** instruction.

When an instance of **C** is created, **A::create()** is called and then **C::create()** is called. **B::create()** is called explicitly by **C::create()**, before the body of the **C::create()** method is executed.

Control Classes

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

Closing a Combo Box List using the Esc Key (PAR 65455)

When a combo box list is open, the Esc key now closes the combo box list.

Notes If the form has a **Button** control with the **cancel** property set to **true** and the open combo box has focus, the Esc key is processed by the combo box; not by the **Cancel** button. The **Cancel** button action occurs only if the combo box list is not open.

The Alt+UpArrow shortcut keys action closes the combo box list when it has focus and the Alt+DownArrow shortcut keys action opens the combo box list when it has focus.

Color of Entities in the JadeEditor Control (PAR 66015)

The "Using the JADE Editor" topic and the **initializeJadeEditor** and **setCurrentSchema** methods in Volume 3 of the JADE Encyclopaedia of Classes now contain a note stating "Call the **initializeJadeEditor** method before you call the **setCurrentSchema** method, to ensure that the form displays entities in the correct color.".

Control Class getControlWindowId Method (NFS 65256)

The **Control** class now provides the **getControlWindowId** method, which returns the Windows identifier that JADE creates and assigns to each control window.

In earlier releases, this Windows identifier of the control was used by some testing tools to locate the control elements involved in the script processing via a Windows API call. It can now also be accessed from your JADE code.

Control Order on Touchscreens (PAR 66097)

When a device is running with accessibility options enabled, JADE changes the z-order of controls within each parent to match the **tabIndex** property value order when the form is created so that screen-reading software such as JAWS or Microsoft Narrator will read the correct text associated with controls. These software utilities assume that the z-order defines the control associations; for example, the label associated with a text box or combo box.

This means, however, that any arrangement of the z-order of the controls within the JADE Painter could be lost unless the **tabIndex** property values match the required arrangement of the controls.

Notes Changing the tab index of controls at run time does not have any impact on the z-order of controls within the parent.

Calling the **Window** class **zOrder** method at run time also overrides the z-order established when the form was created, regardless of whether accessibility is active or not.

When Accessibility is enabled, JADE cannot be sure whether screen-reading software is active. Narrator, for example, does not set a system flag to indicate it is active, and other software can confusingly claim that screen reading is active. This problem has become more visible with the arrival of touchscreen PCs, where accessibility is usually active by default.

As a result, when defining forms where the **zOrder** of controls is established within the JADE Painter, ensure that the **tabIndex** ordering of the affected controls matches the required order, so that the same behavior occurs on PCs regardless of the accessibility state.

Alternatively, add calls to the **Window** class **zOrder** method to establish the z-order requirements at run time.

Disabled Table Cell Text Color (PAR 66229)

If a disabled cell, row, or column has a specified value of the **foreColor** property, the text in a disabled cell is now displayed using that color. The text in a disabled cell is displayed using the Windows disabled text color (which is gray, by default) if the **foreColor** property of the cell is not specifically set using the **foreColor** property values of the cell, row, or column.

JadeRichText Control readOnly Property (PAR 66107)

In earlier releases, double clicking on a URL in a **JadeRichText** control did not open the URL if the **readOnly** property of the control was set to **true**. The code that prevented the double-click action on a hyperlink in a read-only **JadeRichText** control was designed to prevent the user from opening an embedded object in **readOnly** mode.

This has now been changed so that the double-click action is ignored only if an object is selected. Double-clicking a hyperlink in a read-only **JadeRichText** control processes the URL request (that is, the control can be tabbed to, respond to user mouse actions, and text can be selected, but keyboard input is ignored).

MultiMedia Control Class (NFS 65385)

The **MultiMedia** control class now provides an alternative version that uses .NET, enabling you to access new style media files such as an MPEG Layer-4 Audio (MP4) file.

The **MultiMedia** class now provides the **useDotNetVersion** property, which has a Boolean value. This property, which is available at development and run time, defaults to **false**. When the value is:

- **false**, the control operates the same as it did in earlier versions of JADE (that is, it uses the older-style MCI control)
- **true**, the control creates a Microsoft WPF MediaElement control that is used to play the media files

When using the .NET control style, the JADE **MultiMedia** control mostly operates in the same way as the older style, with the following exceptions.

- If the play bar of the control is shown, it is drawn using WPF entities and it has a different appearance.
- If the value of the **showMenu** property is **true**, the menu that is displayed when the menu button is clicked shows the **View**, **Volume**, and **Speed** menu commands but not the **Copy**, **Configure**, and **Command** menu items.

  In addition, the menu now provides the **Close** command, which closes the medium that is being shown.
- The **mediaName** property can be a URL (for example, http://hostName/images/introduction.mp4).
- The .NET style *may* not handle older style media files.
- The following **MultiMedia** class properties and methods are not available, and generate exception 1068 (*Feature not available in this release*).
  - eject
  - newFile
  - playFromTo
  - playReverse
  - record
  - save
  - sendString
  - showRecord (set property)
  - timeFormat (set property)

- The methods listed in the following table return fixed values.

Method	Fixed Value
canEject	false
canPlay	true
canRecord	false
canSave	false
timeFormat	null
usesFiles	true

- The **openDialog** method includes MP4-type files in the files that are displayed.
- Media attribute values position (obtained by calling the **getEndPosition**or **getStartPosition** method) and length (obtained by calling the **getLength** method) are not available until the medium has been opened and the **notifyMedia** or **notifyMode** event has been received.

Offsetting Text in a Text Box (NFS 65840)

The **TextBox** control class now provides the **textOffset** property, which specifies in pixels the left and right margin offsets for text displayed in a text box. This can be used to improve the appearance of the displayed text.

The default value is zero (**0**). An exception is raised if the value is set to a negative value.

When the value is greater than zero, the text in the text box is displayed the specified number of pixels from the left and right of the text box client area, to create a left and right margin where text is not displayed.

Note The specified value is ignored if the effective text area is less than 5 pixels wide (that is, the text is too small for the margin to be applied).

Rich Text Control wantReturn Property (NFS 66371, JEDI 170)

JADE has now added the **wantReturn** boolean property to the **JadeRichText** control. This property, which can be read or written to at any time, specifies whether carriage returns are passed to the rich text control.

When this property is set to the default value of **false**, entering a carriage return while the rich text control 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 rich text control.

When the **wantReturn** property is set to **true** and the rich text control has focus, a carriage return is always passed to the rich text control and any default button is unaffected. (This property is ignored in web-enabled forms.)

Text Box Hint Text and Color (NFS 65829, JEDI 117)

The **TextBox** control now provides the properties listed in the following table.

Property	Value	Default	Description
hintText	String	An empty string	Text that is displayed in an empty text box as a hint
hintBackColor	Integer	#80000000 (use the defined **backColor** value)	Background color of hint text
hintForeColor	Integer	#80000000 (use the defined **foreColor** value)	Color of the hint text

If the value of the **hintText** property is:

- Empty, these properties have no impact on the text box display
- Not empty, the specified hint text is displayed in the text box when the value of the **text** property is empty

When the value of the **hintText** property is displayed, the:

- **text** and **selText** properties both continue to return an empty string
- Text cannot be selected or deleted, and the cursor is always positioned at the beginning of the text

As soon as text is entered or pasted into the text box, the hint text is removed and replaced with the specified or pasted text. If the entire text is removed, the hint text is displayed again.

When hint text is displayed, the:

- Back ground of the text box is drawn using the value of the **hintBackColor** property except when it is **#80000000**, in which case the **TextBox** control **backColor** property value is used.
- Text of the text box is drawn using the value of the **hintForeColor** property except when it is **#80000000**, in which case the **TextBox** control **foreColor** property value is used.

The values of the **dataType**, **case**, and **selectionStyle** properties are ignored when the hint text is displayed; for example, hint text can be displayed for a numeric field. The hint text, which is always displayed in the case of its defined string, can never be selected. In addition, for a password text box, the hint text is displayed as clear text (that is, it is not displayed using asterisk (*) characters).

When the value of the **hintBackColor** or **hintForeColor** property is **#80000000**, the value of the respective **hintBackColor** or **hintForeColor** property in the JADE Painter Properties dialog is displayed as **Use backColor Value** or **Use foreColor Value**, with an image at the left of the text displaying the current value of the respective **backColor** or **foreColor** property. The combo box drop-down list of colors includes an entry for **Use backColor Value** or **Use foreColor Value** that sets the property value to the default.

Note If the values of the **hintBackColor** and **hintForeColor** properties are both zero (**0**), the default values of **#80000000** (that is, transparent) are used instead, to allow for the handling of existing text box definitions from earlier releases.

Creating an RPS Database from a Backup of the Primary (PAR 65995)

A separate topic has now been created in Chapter 2 ("Creating the RPS Database") of the *JADE Synchronized Database Service (SDS) Administration Guide*. The topic heading and the text explicitly state that you can create an RPS node from an online quiesced or an offline backup of the primary.

Creating JSON using the JadeJson Class (NFS 65821, JEDI 130)

When creating JSON from an object using the **JadeJson** class, the type tag is now inserted only when the class is not specified; that is, the **"__type:<class name>"** (Microsoft style) and **"$type:<class name>"** (NewtonSoft style) tags are output only when the object is a subclass of the specified type. For example, if a reference property is defined to be a **Customer** and the reference is to a subclass **SpecialCustomer**, the tag is inserted.

Similarly, if a collection is encoded to JSON and a member of the collection is a subclass of the membership type, the tag is inserted.

Database Reorganization Status Methods (PAR 65686)

The **Schema** class now provides the following methods that identify whether a database has an incomplete reorganization pending.

- **reorgInProgress**, which returns **true** if a reorganization is currently "actively" in progress; otherwise it returns **false** (that is, it returns **false** if the reorganization is in dual-update mode and waiting for the transition).

- **reorgIsWaitingForTransition**, which returns **true** if in dual-update mode and waiting for the transition; otherwise it returns **false**.

DateTimeOffset .NET Component Library Support (NFS 66339)

The .NET **DateTimeOffset** type and the JADE **TimeStampOffset** type are now directly mapped when a .NET library is imported into JADE.

As these types are stored slightly differently, the following conversion takes place to maintain consistency.

- The .NET **DateTimeOffset** type **DateTime** value represents a local time (UTC time + Offset).
- The JADE **TimeStampOffset** type **TimeStamp** value represents a UTC time.

Because of this, when converting from the JADE **TimeStampOffset** type to the .NET **DateTimeOffset** type, the **DateTimeOffset** is derived by applying the offset to the displayed **TimeStampOffset** type UTC time value; for example:

```
TimeStampOffset: 09 October 2018, 21:00:00 +1300

DateTimeOffset: 10 October 2018, 10:00:00 +1300
```

endJade Non-GUI Client Argument

Support for the **endJade** argument is maintained to provide backward compatibility for existing applications. It is recommended that your applications use the **startAppParameters** and **endAppParameters** arguments.

Specify the optional **endJade** argument if you want to specify your own arguments (that is, user-defined command line arguments) in addition to the JADE-specific ones that are required. If you specify this argument, it must be the last of the JADE arguments before the first of your own arguments.

Error when Committing a Transaction (PAR 65012)

JADE no longer permits committing a transaction while executing a constructor or destructor for a persistent instance, because doing so could result in a logically inconsistent transaction.

If a persistent transaction is committed while executing a constructor or destructor for a persistent instance or a shared transient transaction, or a shared transaction is committed while executing a constructor or a destructor for a shared transient instance, JADE now raises exception 1187 *(Attempt to change transaction state within a constructor or destructor)*. This exception is also raised if an exception handler armed in a constructor or destructor aborts the transaction and attempts to return control within the constructor or destructor by returning **Ex_Resume_Next**.

To prevent this exception being raised, change your application code to handle the committing of transactions appropriately.

Extracting and Loading Definition Files in DDX Format (NFS 66195)

The **.ddb** form and data definition file is output using a specialized format that is difficult for humans to read and to compare the file against other versions of the file to detect what has been changed. The file also contains JADE oids, which makes comparison impossible.

You can now extract and load form and data definition files in the XML Device Data Exchange (DDX) format instead of the default existing Device-Dependent Bitmap (DDB) format. This XML format does not include JADE oids. All entities are identified by name and by their position in the XML object hierarchy. You can compare the DDX file to another version of the file, to identify what has changed between the two versions.

The following changes have been made to the load and extract processes and to the JADE development environment.

- All load dialogs that contain a *forms-name* control for the loading of schema definitions (for example, the **Forms File Name** text box) for the name of the .**ddb** file type now allow the .**ddx** file type.
- When a definitions file is to be loaded, the file header is examined to determine if it is a JADE XML DDX format (.**ddx**) file. If it is a DDX format file, it is loaded using new XML handling routines. If the definition file is *not* a valid .**ddx** file, it is assumed to be the older style .**ddb** type file and loaded accordingly.

  The first line of a .**ddx** file must have the **<?xml...** header. The format of the second line is:

  ```
  <schema name="schema-name" JadeVersionNumber="JADE-version"
  JadePatchNumber="patch-number" CompleteDefinition="true|false">
  ```

  The following is an example of the first and second lines of the XML file.

  ```
  <?xml version="1.0" encoding="utf-8"?>
  <schema name="CalculatorSchema" JadeVersionNumber="18.0.01" JadePatchNumber="0"
  CompleteDefinition="true">
  ```

  The schema name, which specifies the schema the information is for, must be included.

  The **JadeVersionNumber** tag identifies the version of JADE that was used to produce the file.

  The **JadePatchNumber** tag specifies the patch number to use for the load; otherwise the current patch number is used.

  The **CompleteDefinition** tag, which must be present, specifies whether the file is a complete definition for the whole schema or it is a partial schema. If the value of the **CompleteDefinition** tag is **true**, any existing entities not included in the file are deleted.
- The **Extract Options** sheet of the Extract dialog now contains the **Extract Forms/Data as XML (ddx file)** check box. If the check box is not checked, the forms and data definition file format type defaults to **ddb**, and an older-style DDB file is produced. Check this check box if you want the forms and data definition file type to default to **ddx**), and a newer-style XML file produced.

  The value of this check box defaults to the value of the new **Use DDX style (xml format) as Default instead of DDB** check box on the **Schema** sheet of the Preferences dialog. By default, this check box is unchecked; that is, schemas are extracted as .**ddb** files.
- The value of the **ddbFile** argument in the **jadloadb** batch Schema Load utility and non-GUI **JadeSchemaLoader** application command lines when running from a command script can be the fully qualified name of the forms and data definition .**ddb** or .**ddx** file; for example:

  ```
  jade.exe schema=RootSchema app=JadeSchemaLoader path=d:\jade\system
  ini=d:\jade\myjade.ini AppServer=MyAppServer AppServerPort=1234
  startAppParameters schemaFile=d:\jade\scm\test.scm ddbFile=d:\jade\scm\test.ddx
  dontSaveSources=true deletePropertiesIfAbsent=true
  loadStyle=onlyStructuralVersioning replayableReorg=true
  compileUnchangedMethods=true endAppParameters
  ```

  For any batch load or extract process, specifying a forms and data definition file extension of .**ddx** results in the newer-style XML file being loaded or extracted; otherwise a .**ddb** file is processed.
- The value of the **ddbFile** argument in the **jadclient** non-GUI application when extracting a schema from a command script can be the fully qualified name of the forms and data definition .**ddb** or .**ddx** file; for example:

  ```
  jadclient path=e:\jadesystems\test\system ini=c:\jade\system\jade.ini
  app=JadeBatchExtract schema=JadeSchema server=singleUser startAppParameters
  File d:\temp\delta.scm d:\temp\delta.ddx d:\temp\param.unl DbTestSchema
  delta=DeltaTwo
  ```
- In earlier releases, extracting a **Window** or **JadeHTMLClass** subclass prompted you for the location and file name of the class (.**cls**) and used the same name and location for the .**ddb** file.

The Extract dialog is now displayed, so that you can also specify the type of the forms and data definition file (that is, .**ddb** or .**ddx**), as well as the file location and names.

- Extracting an exported package using the JADE development environment now also displays the Extract dialog, so that you can specify the type of the forms and data definition file (that is, .**ddb** or .**ddx**) to produce.
- When extracting a schema with an RPS mapping and the DDX format style is selected, a separate .**ddbx** file is produced as for a DDB extract but the file contents are in XML format.

Note When extracting a schema metadata file and form and data definitions file with the DDX file format, the **Global** class of the schema is extracted but the data is ignored when the schema is loaded. (For a DDB format file, the **Global** class is not extracted.)

The global information in the DDX file is ignored for consistency and because it creates a situation where the data cannot be loaded until a reorganization is performed if the **Global** class is marked for reorganization.

JADE Development Environment

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

Adding a Map File from the Define Class Dialog (NFS 64830)

The **Class** sheet of the Define Class dialog now provides the **Add Map File** button, to enable you to add a new map file for the current schema. The File Dialog is then displayed, to enable you to define your new map file.

When you have specified the new map file and clicked the **OK** button on the File Dialog, the class map file is displayed in the schema hierarchy of the Class Maps Browser and is selected in the **Map File** combo box on the **Class** sheet of the Define Class dialog.

Notes The map file addition is an independent update that remains defined even if you cancel the class definition.

If the **Map Files** combo box is disabled, the **Add Map File** button is also disabled. It is disabled if the class has a predefined map file or the class has instances.

Adding a New Class to a Refactored Method (NFS 64111)

When editing a JADE method, you can now create a new class without leaving the editor pane, even if the current method has been changed and not saved.

» **To add a new class**

1. In the editor pane, select the **Add Class** command from the Refactor submenu of the Edit menu. The Define Class dialog is then displayed. For details, see "Defining Your Own Classes", in Chapter 3 of the *JADE Development Environment User's Guide*.
2. As the **Object** class is selected by default in the **Subclass of** combo box on the **Class** sheet of the Define Class dialog, you must select the class that you require as the superclass of the new class.

The class is added to the current schema and is displayed in the Class List (you may have to expand the appropriate superclass). The currently selected class, property, and method remain selected.

Alternating Colors for List Box and Table Rows (NFS 65216)

JADE now provides the following four **Integer** value properties that control whether the rows in list boxes and tables are drawn with alternating background colors.

- **ListBox** class **alternatingRowBackColor** property, which can be set in the JADE Painter and in runtime logic. The default value is **Azure** (a new global constant in the **ColorConstants** category).

- **ListBox** class **alternatingRowBackColorCount** property, which can be set in the JADE Painter and in runtime logic. The default value is zero (**0**).
- **JadeTableSheet** class **alternatingRowBackColor** property, which can be set only by runtime logic. The default value is **Azure** (a new global constant in the **ColorConstants** category).
- **JadeTableSheet** class **alternatingRowBackColorCount** property, which can be set only by runtime logic. The default value is zero (**0**).

For a **ListBox** control, if the value of the **alternatingRowBackColorCount** property is:

- Less than or equal to zero (**0**), the background color of each list box entry defaults to the value of the **backColor** property of the **ListBox** control. The **alternatingRowBackColor** property value is ignored.
- Greater than zero (**0**), for each **alternatingRowBackColorCount** property visible entry, the background color defaults to the value of the **alternatingRowBackColor** property.

For example, if the count is **2**, the first, third, fifth, and so on visible entries in the list default to the value of the **backColor** property of the **ListBox** control, while the second, fourth, sixth, and so on visible entries default to the value of the **alternatingRowBackColor** property.

If the value of the **itemBackColor** property of the list item is specifically set and it is not **#800000000** (that is, transparent), the default value of the list box item is ignored and the value of the **itemBackColor** property is used.

For a **Table** control, if the value of the **alternatingRowBackColorCount** property is:

- Less than or equal to zero (**0**), the background color of each non-fixed cell defaults to the value of the **backColor** property of the sheet, or of the table itself if the value of the sheet is not specifically set. The **alternatingRowBackColor** property value is ignored.
- Greater than zero (**0**), for each visible **alternatingRowBackColorCount** non-fixed row and non-fixed cell, the background color defaults to the value of the **alternatingRowBackColor** property.

For example, if the count is **2**, the first, third, fifth, and so on non-fixed rows and the non-fixed cells in that row default to the value of the **backColor** property of the sheet, while the second, fourth, sixth, and so on non-fixed rows and the non-fixed cells in that row default to the value of the **alternatingRowBackColor** property.

If the value of the **backColor** property of a cell, row, or column is specifically set and it is not **#800000000** (that is, transparent), the default value of the cell is ignored and the specific value of the **backColor** property is used.

Note that when the list box or table is scrolled, the colors do not move with a row. The color scheme is applied to the rows, starting with the first visible non-fixed row; for example:

```
table1.accessSheet(1).alternatingRowBackColorCount := 2;
table1.accessSheet(1).alternatingRowBackColor := Azure;

listbox1.alternatingRowBackColorCount := 3;
listbox1.alternatingRowBackColor := DarkGray;
```

In addition, the **Window** sheet of the Preferences dialog now contains the **Show Alternating Row BackColor** check box, which is unchecked by default. When you check this check box, tables and list boxes in browsers (for example, a hierarchy browser such as the Class Browser) in the JADE development environment set the value of the **alternatingRowBackColorCount** property to **2**.

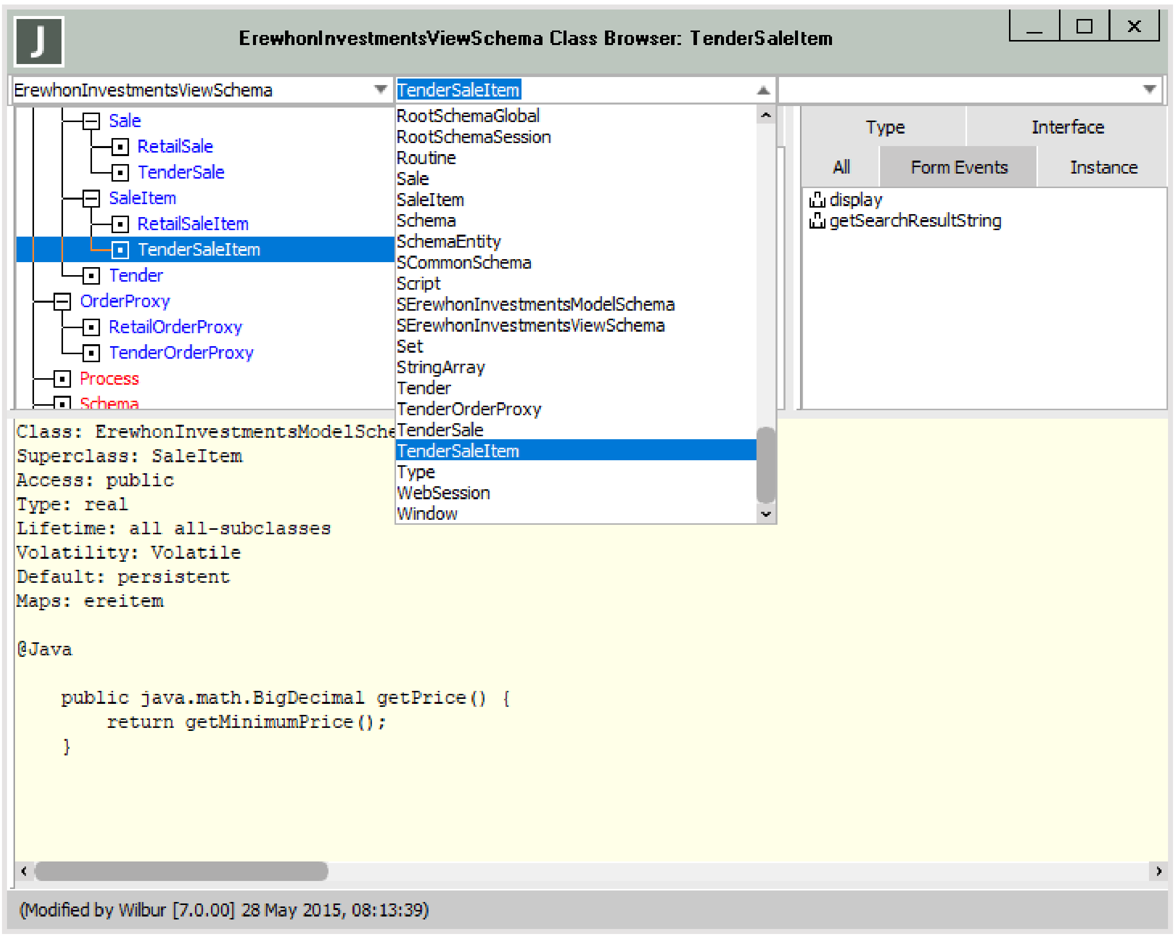
The value of the **alternatingRowBackColor** property value defaults to **Azure**, but you can change it by selecting the required color displayed in the second column of global color constants in the table at the upper-left of the **Window** sheet of the Preferences dialog. These values are saved in your user profile when you extract your user preferences to a file and they are restored when you reload your preferences.

Note The **alternatingRowBackColor** and **alternatingRowBackColorCount** properties have been added to the **JadeSkinListBox** and **JadeSkinTable** classes. You can set these values in a **ListBox** and **Table** skin using the functionality provided by the **JadeSkinMaintence** form. If the value of the **alternatingRowBackColorCount** property is zero (**0**), the value of the **alternatingRowBackColor** property is ignored and does not apply.

The skin settings are also ignored if logic specifically sets the value of the **alternatingRowBackColorCount** property of a list box or table to a value greater than zero (**0**). See also "Table Row Alternating Colors (NFS 65216)" and "Specifying Some Skin Table Colors (NFS 65152)" under "JadeSkinControl Subclasses", later in this document.

Alternative Navigation Mechanism (NFS 65350)

Each hierarchy browser (for example, the Class Browser) now provides a navigation bar, with drop-down combo boxes from which you can select a schema, a class, and a method.



```
Class: ErewhonInvestmentsModelSche
Superclass: SaleItem
Access: public
Type: real
Lifetime: all all-subclasses
Volatility: Volatile
Default: persistent
Maps: ereitem

@Java

    public java.math.BigDecimal getPrice() {
        return getMinimumPrice();
    }
```

(Modified by Wilbur [7.0.00] 28 May 2015, 08:13:39)

Selecting a schema, class, or method refreshes the current browser with the selected entity; for example, selecting a schema refreshes the display to that schema (as though a new browser was opened) and selecting a class selects the class (as though a class was selected using F4).

The display is updated when the combo box is closed and a list entry has been selected. This enables you to use the arrow keys to scroll through the entries and enter text to refine the list before clicking the required entry or pressing Enter.

The navigation bar is hidden, by default. Toggle the display of the navigation bar by checking or unchecking the **Show Navigation Bar** check box on the **Browser** sheet of the Preferences dialog. The setting of this check box is saved in your user profile and can be extracted to a preferences initialization file by using the existing Export Preferences functionality provided on the **Miscellaneous** sheet of the Preferences dialog.

Tip You can also toggle the navigation bar display by using the View menu **Show Navigation Bar** command.

The navigation bar combo boxes, from left to right, are as follows.

- When the combo box on the left of the browser is dropped down, it lists the name of every schema defined in your system, including versioned schemas (which are displayed with the versioned background color of your user preference).

  Selecting a schema changes the view to classes in the selected schema, with the **Object** class selected.

  The text box portion of the combo box usually displays the name of currently selected schema.

- When the middle combo box is dropped down, it lists in alphabetical order the name of every class defined in the selected schema.

  Selecting a class in the list is equivalent to clicking on the class in the Class List of the browser.

  The text box portion of the combo box usually displays the name of the currently selected class.

  The list does not include any superschema classes not marked as copy classes in the currently selected schema. To access such classes, press F4.

- When the combo box at the right is dropped down, it lists in alphabetical order the name of every method in the currently selected class.

  When the class is a form, the list includes all event methods defined for the form, menus, and all controls. The event methods display the full name (for example, **cmbCustomers_click**).

  This list does not include the names of any superclasses, regardless of the setting of the View menu **Show Inherited** command.

  Selecting a method in the list displays the selected method. If the method is in the currently displayed method list, that method entry is selected. If the method is *not* in the currently displayed list, the **All** folder of the Methods List is displayed and the method is selected in the methods list of the combo box.

  If the method is an event method, the associated form, control, or menu item in the Properties List is selected (and the **All** folder in the Properties List is displayed).

  If the method is not an event method, the properties selection is unaffected(that is, the **<methods>** entry is no longer displayed in the Properties List).

If one of the combo box entries is selected and the current method display has been changed but not saved, you are prompted to save or discard the changes, or to cancel the action.

Each combo box enables you to enter text. Changing the combo box text causes the display of only entries that contain the specified text somewhere in the entity name.

The search process handles uppercase searches if all text entered after the first character is uppercase. The list then includes all entries in which the first character of the entity name matches the first entered character and the entity name has uppercase characters that match the order of the remaining entered uppercase text. For example, entering **sNT** in the method combo box at the right of the navigation bar include method names **searchNameTextOnly** and **someNumberedTest**.

Note Because the same form can be used to access different schemas, the browser history in the displayed History menu includes the schema name if the entry is not for the currently selected schema.

If the accessed schema was versioned, any history menu list entry for the versioned schema (for example, the schema or a class name) is preceded by an asterisk (*) character.

Application Browser

This section describes the Application Browser changes in this release.

Application Browser Methods Display (NFS 65359)

If the **initialize** or **finalize** method is not set for an application but it is defined in the application or any superclass other than the RootSchema **Application** class, the Application Browser now displays that method in the respective **Initialize Method** or **Finalize Method** column of the Application Browser.

This enables you to see from the Application Browser the default **initialize** and **finalize** methods that will be invoked when running the application.

Tip Double-clicking on a method name in the **Initialize Method** or **Finalize Method** column displays the method source for that method.

Application Browser Table Sorting (NFS 64485)

You can now sort the table in the Application Browser by clicking on the heading of a column to sort by that value (for example, **Application Type**). If the column is not the **Name** column, the table is sorted using the selected column and then the **Name** column as the second sort column.

Clicking on the existing sorted column heading toggles ascending sort order.

Application Initialize and Finalize Methods (NFS 64881)

The **Application** sheet of the Define Application dialog now provides the **Show Super Class Methods** check box, which you can check if you want applicable methods from the current schema and all superschemas displayed in the **Initialize Method** and **Finalize Method** combo boxes. When this check box is unchecked, applicable methods from the current schema are displayed in the combo boxes.

This check box is initially unchecked for a new application definition or when you are changing an application and the **Initialize Method** and **Finalize Method** combo boxes are empty or the methods are defined in the current schema. If an application being changed has an **initialize** or **finalize** method that is defined in another schema, the check box is checked and disabled.

If you check this check box to toggle the checked status, the **Initialize Method** and **Finalize Method** combo boxes are reloaded according to the new status.

When you select an **initialize** or **finalize** method from a superschema, the check box is disabled so that you cannot uncheck the check box, which would prevent the display of the currently selected method.

AutoComplete Feature

This section describes the AutoComplete feature changes in this release.

AutoComplete and Editor Pane Colors (NFS 65529, JEDI 52)

The **Editor** sheet of the Preferences dialog now provides the following component default colors. (You can change the colors specified in the following list items, to meet your requirements.)

- **Method (AutoComplete)**, which defaults to brown in **AutoComplete** list boxes for *instance* methods
- **Type Method (AutoComplete)**, which defaults to faded blue in **AutoComplete** list boxes for *type* methods
- **Property (AutoComplete)**, which defaults to light blue in **AutoComplete** list boxes
- **Additional Selections**, which defaults to a background of lime green, to highlight other text in the method that matches the current selection

Display of Dictionary Keys (NFS 65268)

The AutoComplete feature now displays the required dictionary keys when a method has a **KeyType** parameter. The displayed signature of the method is revised to display the keys of the referenced dictionary instead of the **KeyType** parameter when the cursor is positioned in the method parameters. For example, **dict.getAtKey(keys: KeyType)** is displayed as **dict.getAtKey(name: String; dob: Date)** where the keys of the dictionary are **name** and **dob**.

In addition, when hovering the mouse over the method name, the bubble help display now displays the defined signature followed by **Required method call: *<revised method call with the key parameters>***.

For example:

```
(method) <schema-name><class-name>.getAtKey(keys: KeyType)
```

Required method call:

```
(method) <schema-name><class-name>.getAtKey(name : String; dob : Date)
```

The change also handles implied calls to **Dictionary** class getAtKey methods involving the<*key*>(**[..]**) substring operator notation. In this case, the method signature of the getAtKey method is displayed showing the keys when the cursor is positioned inside the <*key*> field.

Note, however, that if the collection type is generic (for example, **MemberKeyDictionary**), the **KeyType** parameter cannot be evaluated because the actual collection type is not known at compile time. For example, the following code fragment shows **KeyType** in the signature in AutoComplete.

```
method1(coll: MemberKeyDictionary);
begin
    coll.getAtKey("name", "value");
```

Editor Pane Keyword Recognition (PAR 65777)

JADE now separates the keywords that are valid in a method from those that are valid only in a schema file (for example, **typeDefinitions**, **_encryptedSource**, and **peerOf**) so that the editor pane and AutoComplete feature now use the list valid for the context; that is, method keywords rather than schema file keywords.

Schema keywords are used only when displaying a schema file when a schema load fails because of syntax errors in the schema file being loaded.

Enhancements and Method Error Analysis (NFS 65977)

When the AutoComplete feature is in use, the JADE editor can now perform error analysis of the method definition as you modify the method, which makes you aware of issues before you compile the method.

When an error is detected, the text in error is decorated with a red wavy underline symbol. Hovering the mouse over such text displays a bubble help entry with a brief description of the error.

This analysis is controlled by the **Perform Method Error Analysis** check box in the Auto Complete group box on the **Editor Options** sheet of the Preferences dialog. (This check box is checked, by default.)

The error analysis does not use the compiler and it runs on the client inside the editor. The analysis detects most issues, including:

- Syntax errors
- Invalid local variables and constant definitions
- Invalid method and parameter options
- Unknown identifiers
- Invalid type comparisons and assignments

- Invalid parameter type passed to a method
- Invalid method parameter count
- Invalid use of parameters according to their declared usage; for example, assignment to a constant parameter
- Invalid calls to an updating method from a non-updating method
- Unmatched block endings and break commands; for example, an **if** instruction without a matching **endif** instruction and **break** instruction not in a loop
- Invalid mixed expression types
- Unmatched parentheses
- Invalid type casts

The analysis does not check the following. (This list may grow in the future as other unhandled cases as identified.)

- Whether the method option is appropriate (except for **updating**); for example, **webService**, **unitTest**, and so on
- The class lifetime requirements for a **create** instruction

---

Note The impact on you should be minimal, with the exception being that the AutoComplete feature requires access to all of the entities referenced in the current page of logic. This may cause a longer lead time to retrieve these entities from the server if AutoComplete has not seen them before. Once those entities have been retrieved and cached on the client, the operation should not cause any further delays. As this functionality is on by default, if performance is ever an issue, turn off the user preference.

Method analysis is ignored for an external function definition and for a Workspace.

The method analysis is performed:

- After the editor pane gains focus and the method has been changed or it has compile errors.
- One second after you have changed a method and have paused typing.
- After you have changed the method using a selection from the **AutoComplete** list box.

.Note that with these changes, the following enhancements have been made to the AutoComplete feature.

- **userFormat** references in logic are now handled
- The list box prompt in some areas of a method declaration (for example, constant values) that were previously not handled are now always displayed
- Package references (for example, parameter types) are now handled
- AutoComplete can now determine the type of an expression and can now offer the correct list box alternatives; for example:

  ```
  (amethodCall() * 56 + 1)
  ```

- Use of **([..])** expressions for **Dictionary** and **ExternalDictionary** types that actually mean a call to the **getAtKey** method display the signature required for the **getAtKey** call and highlight the current parameter that is being edited.
- Use of **([..])** expressions for **Array** and **ExternalArray** types that actually mean a call to the **at** method display the signature required for the **at** call and highlight the current parameter being edited.

Using the History for Selection (NFS 65638)

The Auto Complete group box on the **Editor Options** sheet of the Preferences dialog now contains the **Use History for Selection** check box, which is unchecked by default, which means that the entry selected in the AutoComplete list is always the first name with the prefix matching the text entered in the editor pane. The previous history of selected AutoComplete entries is not used and you must type more of the identifier name to locate the required entry in the list box or select the required entry using an arrow key or the mouse.

When the **Use History for Selection** check box is checked, the last selection made whose prefix matches the text entered in the editor pane is selected by default. If there is no prior history that matches the specified text, AutoComplete operates as though the value of the check box is **false** (that is, it is unchecked).

Bubble Help Display (NFS 65057)

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.

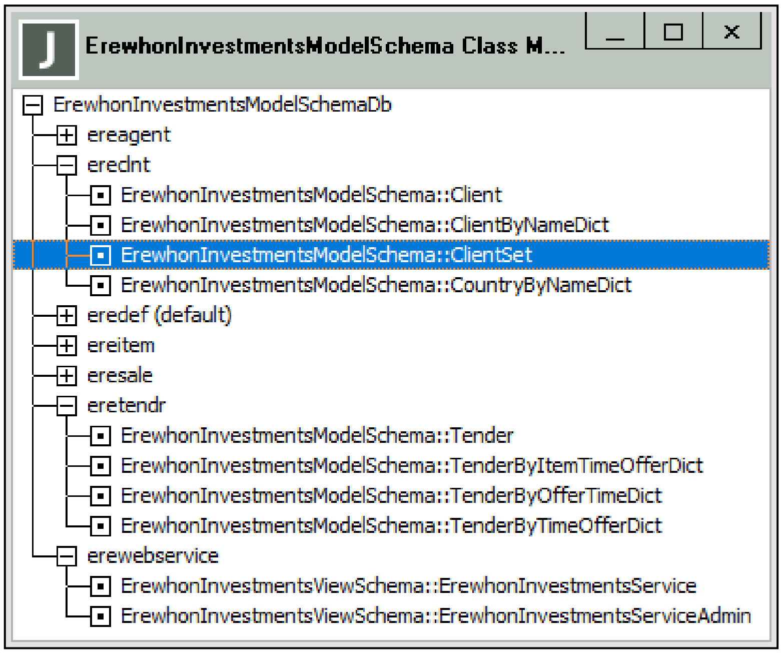
Changing an Application from the Status Line (NFS 64221)

You can now set the current default application under which a Workspace or **JadeScript** method executes from the JADE development environment background window, by clicking on the application name displayed at the far right of the status line. A combo box is then displayed, listing all of the application names for the currently selected schema, with the current default application selected.

Select the application you require as the new current application default. To cancel the combo box display, press the Esc or Tab key, or move focus to another window.

Class Maps Browser (NFS 64862)

The Class Maps Browser now sorts the class names and prefixes each class with the name of the schema in which it is defined, as shown in the following image.



```
ErewhonInvestmentsModelSchemaDb
  ereagent
  erednt
    ErewhonInvestmentsModelSchema::Client
    ErewhonInvestmentsModelSchema::ClientByNameDict
    ErewhonInvestmentsModelSchema::ClientSet
    ErewhonInvestmentsModelSchema::CountryByNameDict
  eredef (default)
  ereitem
  eresale
  eretendr
    ErewhonInvestmentsModelSchema::Tender
    ErewhonInvestmentsModelSchema::TenderByItemTimeOfferDict
    ErewhonInvestmentsModelSchema::TenderByOfferTimeDict
    ErewhonInvestmentsModelSchema::TenderByTimeOfferDict
  erewebservice
    ErewhonInvestmentsViewSchema::ErewhonInvestmentsService
    ErewhonInvestmentsViewSchema::ErewhonInvestmentsServiceAdmin
```

Cleaning Up Unused Variables (NFS 64037)

The Find Unused Variables dialog, accessed from the Methods menu **Unused Local Variables** command, scans the defined local variables in the method in the editor pane and now enables you to remove one or all unused local variables from the source of that method. The **Unused Local Variables** command is disabled if the method is uncompiled or in error.

If there are no unused variables, a message box displays *There are no unused local variables*.

If an unused local variable is found, the Find Unused Variables dialog displays the name of the first unused local variable and the unused variable name is also selected in the method source in the editor pane.

The dialog now has four buttons, as follows.

- **Find Next**, which displays the next unused variable. If there are no more in the current method, the dialog is closed and a message box displays *There are no more unused local variables*.
- **Remove**, which removes the displayed unused variable from the method source. The logic then continues as though you had clicked the **Find Next** button.
- **Remove All**, which removes all unused variables from the method source. The dialog is then closed and a message box lists the names of all variables that have been removed.
- **Cancel**, which closes the dialog. (No previous remove actions are undone.)

Note Any changes made to the method can be undone and the method will be unsaved.

Closing All MDI Windows Confirmation (NFS 65884)

When you select the Windows menu **Close All Mdi Forms** command, a confirmation dialog is now displayed so that you must click **Yes** to proceed with the closing of all MDI forms or **No** to abandon the request.

Comparing Method Source (NFS 64636)

You can now change the size of the merge editor pane, by dragging the (splitter) resize bar. When the Compare Sources window is unloaded, the size of the dock control that contains the merge editor pane is also saved when the **Save Size and Position** check box is checked on the **Window** sheet of the Preferences dialog. For example, if you have dragged the splitter (resize) bar right down to get the merge editor pane out of the way and you close the form, it is not displayed when you reload the Compare Sources window until you drag up the splitter bar up.

Controls in the Class Hierarchy Browser (NFS 65880, JEDI 136)

When a property is clicked in the Properties List of the Class Hierarchy Browser or bubble help for a property and that property is a control or menu instance on a form, additional information is now displayed in the editor pane.

For a control, the display includes the control parent hierarchy within the form and the value of the **caption** (if the control has that property), **left**, **top**, **width**, **height**, **enabled**, and **visible** properties; for example:

```
Name: btnChgCtlPalPic (22)
Class: ClassDefinitionDialog ()
Type: Button
Access: public
Ordinal: 4
non-virtual embedded control

Control parent hierarchy on form:
    ClassDefinitionDialog::Form
        fldDefineClass::Folder
            shtCtrlOptions::Sheet
                grpPainterIcon::GroupBox
```

```
Caption: 'Change...'
Left: 48 Top: 59 Width: 80 Height: 59
Enabled: true
Visible: true
```

For a menu property, the display includes the menu parent hierarchy on the form or that the menu item appears on the menu line of the form, and the value of the menu **caption**, **level**, **enabled**, and **visible** properties; for example:

```
Name: mnuRefactorIDeclareVar (216)
Class: JadeWindow ()
Type: MenuItem
Access: public
Ordinal: 251
non-virtual embedded menu

Menu parent hierarchy on form:
    mnuEditMenu Caption: &Edit
        mnuRefactor Caption: R&efactor

Caption: '&Declare Local Variable'
Level: 3
Enabled: true
Visible: true
```

Copying a Method (NFS 65852)

The **New Name** text box in the Copy Method As dialog now displays the name of the existing method that is being copied, and the entire method name is selected.

Customizing the Layout of Hierarchy Browser Forms (NFS 66119)

The Hierarchy Browser form has been changed so that you can:

- Drag and drop the four major panels (the Class List, Properties folder, Methods folder, and editor pane) into a new layout; for example, you could vertically dock the Class List, Properties folder, and Methods folder on the left of the form, with the editor pane occupying the full height of the right of the form.
- Float each of the four panels.
- Set a changed layout as the default layout for the current schema. Each schema can have its own layout.

  This default layout applies only to new Hierarchy Browser forms opened for that schema.
- Set a changed layout as the default for all schemas. This default layout applies only to new Hierarchy Browser forms that are opened.

  Note The default layout for a specific schema overrides the default layout for all schemas.
- Clear a saved layout for a single schema or the default layout for all schemas.
- Restore the JADE default layout.
- Apply the saved layout for another schema to the current Hierarchy Browser.

To change the layout, use the standard docking control drag and drop features, as follows.

- Left-click the docking control gripper bar for the panel to be moved and then drag the window to the position you require.

  When the mouse moves over a docking control gripper bar, an icon of a hand with a pointing finger is displayed. This is now standard for a dock control that can be dragged.
- When the left mouse button is down, a drag rectangle is displayed in the area where the pane will be

docked.

- To change the orientation (that is, flip vertical / horizontal orientation), press Shift while the left mouse button is down.
- To float the pane, press Ctrl while the left mouse button is down.
- Abort the drag process, by pressing Esc key before you release the left mouse button.
- To complete the position move, release the left mouse button.
- Use the panel size dividers to resize adjacent panels to meet your requirements.

The Hierarchy Browser View menu now provides the **Layout** command, which has a submenu containing the commands listed in the following table.

Menu Command	Description
Apply Layout	Has one or more submenu commands, as follows. ■ **Default JADE Layout**, which restores the default JADE layout. This command is disabled if the JADE layout is already the default. (Default means that the panels have not been moved, but they may have been resized.) ■ *schema-name*, which are the names of any layouts that have been saved for a schema. Clicking one of these commands applies that layout to the current Hierarchy Browser form.
Clear the Default Layout	Discards the default layout saved for all schemas. This does not affect any displayed layouts. The command is disabled if there is no default layout for all schemas.
Clear the Default Layout for Schema *current-schema-name*	Discards the default layout for the current schema. This does not affect any displayed layouts. The command is disabled if there is no default layout for the current schema.
Save as Default Layout	Saves the current layout as the default layout for all schemas. It does not affect the layout of any open forms. The command is disabled if the current layout is the default JADE layout.
Save as Default Layout for Schema *current-schema-name*	Saves the current layout as the default layout for the current schema. It does not affect the layout of any open forms. The command is disabled if the current layout is the default.

Note the following:

- A panel can be docked only into its original form.
- If you have checked the **Save Windows** check box on the **Exit** sheet of the Preferences dialog, the current layout where the panels have been moved is saved and restored, regardless of your setting of the **Save Size and Position** check box on the **Window** sheet of the Preferences dialog.
- If the layout was the JADE default, it is restored using the JADE Default layout.
- If you have unchecked the **Save Windows** check box on the **Exit** sheet of the Preferences dialog (that is, the value is **false**), any altered layouts are lost. When you open a new Hierarchy Browser form and there is a default layout for that schema or for all schemas, that layout applies.
- The layouts are not saved when you export your user preferences.

Displaying Interface Methods Only (NFS 64656)

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

Clicking on the **Interface** tab displays only interface methods. (Interface methods are still displayed in the **All** sheet along with any other defined methods.)

The list of interface methods is displayed as a two-level hierarchy. The first level is the name of each interface implemented by the class, and the second level is list of methods implemented for that interface. Clicking on the interface name item is ignored (that is, it is disabled).

The caption of the tab of the first folder varies, depending on the entities that are displayed. The caption is displayed as:

- **Form Events**, when a *form* property entry is clicked
- **Events**, when a control property is clicked
- **All**, in all other cases

See also "Method Folders for Form and Control Events (NFS 65485)", later in this document.

Dragging and Dropping Files (NFS 65001)

You can now drag and drop a file in the following cases.

- Dropping one or more files onto the **New Workspace** or **Open Workspace** button on the JADE development environment toolbar opens a new workspace for each file that is dropped.

  Any folders that are dropped are ignored.

- Dropping one or more files onto the **Load** button on the JADE development environment toolbar opens the Load Options dialog and populates the **Schema File Name** text box with the file or files. If you drop:

  - A single file with a **.mul** file type, the **Load Multiple Schemas** check box is checked.
  - More than one file, the **Load Multiple Files** check box is checked and all of the dropped file names are displayed in the **Schema File Name** text box, so that you can add more files, if required.

  Dropping a folder is rejected.

- Dropping one or more files onto the Schema Browser list box opens the Load Options dialog and populates the **Schema File Name** text box with the file or files. If you drop:

  - A single file with a **.mul** file type, the **Load Multiple Schemas** check box is checked.
  - More than one file, the **Load Multiple Files** check box is checked and all of the dropped file names are displayed in the **Schema File Name** text box, so that you can add more files, if required.

  Dropping a folder is rejected.

- Dropping a file onto the Workspace window displays the contents of the file in the workspace, replacing any previous content. If the previous workspace was unsaved, you are first prompted to save or discard the current workspace, or to cancel the display of the dropped file.

  Dropping multiple files is rejected.

See also "Load Options Dialog Files Drag and Drop Supported (NFS 64897)", later in this document.

Editor Pane

This section describes the editor pane changes in this release. (See also "AutoComplete and Editor Pane Colors (NFS 65529)", earlier in this document.)

Editor Pane Zoom Functionality (NFS 65473)

The editor pane now enables you to zoom in or out using the Ctrl key and the mouse wheel, as follows.

- Ctrl+ scrolling the mouse wheel forward increases the zoomed size of displayed editor pane.
- Ctrl+ scrolling the mouse wheel backward decreases the zoomed size of the displayed editor pane.
- Ctrl+ clicking the middle mouse button (that is, pressing the mouse wheel) restores the editor pane to the normal (default) size.

Selecting and Highlighting Text (NFS 65503)

The Scintilla open-source editor version used by JADE for the **JadeTextEdit** control and its **JadeEditor** subclass control has been upgraded to Scintilla version 3.7.4, which provides additional functionality, including:

- Rectangular selection

  You can now make a rectangular selection, by pressing:

  - Alt key when dragging the mouse
  - Shift+Alt+ arrow keys

  You can copy the rectangle selection area to the Windows clipboard or to the JADE editor clipboard. Pasting the text in the clipboard adds the text at the selected position and starts a new line for lines 2 and greater.

  You can also use this functionality to add tab characters into multiple lines, by pressing Shift+Alt+↓ to select the lines (without selecting any characters) into which the tab characters are added and then pressing Tab to enter tabs on each of the selected lines.

  Similarly, to remove tabs, press Shift+Alt+↓ to select the lines (without selecting any characters) from which the tabs are removed and then press Backspace, to move text back to the prior tab position.

- If no text is selected and the cursor is over an identifier, all matching occurrences of that identifier (full-word and case-sensitive) are highlighted.

  When text is selected, the result is determined by the value of the new **Only highlight whole words matching selection** check box on the **Editor Options** sheet of the Preferences dialog. If the check box is:

  - Unchecked (the default value), any other text matching the case-sensitive selection is also highlighted using the **Additional Selections** color specified on the **Editor** sheet of the Preferences dialog (lime green, by default), unless the selection contains only *space-type* characters; for example, selecting the word **to** highlights any occurrence of **to** in the editor pane.
  - Checked, any selection causes other text matching text in the editor pane to be highlighted if the selection is a full-word identifier and there are other occurrences of that word in the editor pane. This match is case-sensitive and full-word; for example, selecting the word **caption** highlights any other occurrence of the full word **caption**.

Exported Package Browser References Command (PAR 65812)

The Packages menu **References** command in the Exported Package Browser is now enabled for interface methods and constants.

Highlighting Methods in Lists of Methods (NFS 65865, JEDI 113)

The method lists in the following forms now provide the **Highlight Method Item Background** command in the Methods menu.

- Methods View Browser
- Method Status List Browser
- Unreferenced Methods Browser
- References Browsers
- Implementor Browsers
- Messages Browser
- Global Find/Replace Results Browser

Clicking the **Highlight Method Item Background** command in the Methods menu displays the following submenu commands.

- Clear Highlighting
- Blue
- Gray
- Green
- Red
- Yellow

You can use the five color commands to set the background color of the selected method list item or items; for example, as a reminder that more work on the selected method or methods is required. The color-related menu commands show the background color that will be used.

Clicking a color sets that color as the background color of the current selected method list item or items.

Select the **Clear Highlighting** command, to clear the highlighting color from the selected item or items. The command is disabled if no highlight has been assigned to the currently selected method list item or items.

Note This highlighting applies only while the current form is open; it is lost when the form is closed.

Highlighting Rounded and Square Bracket Pairs (NFS 65936, JEDI 121)

The JADE editor now highlights parentheses (**()**) and bracket (**[]**) pairs when editing a JADE method or schema file when the cursor is positioned before the starting or closing **()** parenthesis or **[]** bracket character, making it quicker to resolve the omission of the closing parenthesis or bracket.

The background of the two parentheses (**()**) and bracket (**[]**) bracket characters is colored using the **Additional Selections** value specified on the **Editor** sheet of the Preferences dialog. The default value is bright green.

If the cursor is positioned before a parenthesis or bracket character that does not have a matching starting or closing bracket, the background of the character is highlighted in red. (There is no user preference available for this.)

Interface Toolbar Button (NFS 64711)

The main development environment window browser toolbar now contains the stylized **I** (Browse Interfaces) toolbar button, positioned between the **P** (Browse Primitive Types) and **M** (Browse Maps) toolbar buttons.

Clicking the **Browse Interfaces** toolbar button displays the Interface Browser for the currently selected schema.

Load Options Dialog Files Drag and Drop Supported (NFS 64897)

The Load Options dialog now supports the dragging and dropping of files on the **Schema File Name** and **Forms File Name** text boxes.

Dragging and dropping a file on the **Schema File Name** text box has the following effect.

- If a single .**mul** file is dropped on the text box, the **Load Multiple Schemas** check box is checked and the text box displays the name of the multiple schemas file. (The file must be of type .**mul**, for this action to be recognized.)
- If a single schema file is dropped on the text box and the **Load Multiple Schemas** check box is unchecked, the name of the dropped schema file replaces any file that was already displayed in the text box.
- If a single file is dropped on the text box and the **Load Multiple Files** check box is checked, the file is added to the list of file names in the **Select Multiple File Names** text box.
- If multiple form and data definition (.**ddb** or .**ddx**) or method (.**mth**) files are dropped and the **Load Multiple Files** check box is unchecked, the **Load Multiple Files** check box is checked, any previously selected files are cleared, and all of the dropped file names are displayed in the **Select Multiple File Names** text box.
- If multiple files are dropped and the **Load Multiple Files** check box is checked, the files are added to the list of file names displayed in the **Select Multiple File Names** text box.

Dropping a single form and data definition (.**ddb** or .**ddx**) file on the **Forms File Name** text box displays that file name, replacing any previously displayed forms definition file or files. Dropping multiple files on the **Forms File Name** text box in the same action is rejected.

Notes You can drop only files on the **Schema File Name** and **Forms File Name** text boxes; that is, dragging and dropping a folder is rejected.

The file types are not verified, as it is your responsibility to do so.

Method Folders for Form and Control Events (NFS 65485)

The Methods List of browser windows can now contain the **Form Events**, **Cntrl Events**, and **Menu Events** folders in addition to the **All**, **Instance**, **Type**, and **Interface** folders described in "Displaying Interface Methods Only (NFS 64656)", earlier in this document. This change makes it easier to access event methods and non-event methods without having to select a different property list box entry.

By default, the **All** folder is displayed; that is, all instance, interface, and type methods are displayed. Event methods are not included.

Clicking on the **Form Events**, **Cntrl Events**, or **Menu Events** tab displays only form, control, or menu item property event methods, respectively.

The **Form Events** and **Cntrl Events** tabs are hidden if the class selected in the Class List is not a **Form** subclass. In addition, the **Cntrl Events** tab is hidden if the currently selected property is not a control or menu instance.

Note The **<methods>** and **<form>** entries are no longer added to the Properties List for a **Form** subclass.

To edit a:

- Form's event methods, click the **Form Events** tab in the Methods List and then select the required event method.
- Control or menu item event method, click the associated property in the Properties List so that the **Cntrl Events** or **Menu Events** tab is displayed in the Methods List. Click the tab to display the associated sheet, and then select the required event method.

  This sheet remains visible while the associated property remains selected.

You can switch between the method tabs without affecting the property that is selected and you can edit non-event methods, form event methods, and control or menu item event methods by changing the selected sheet in the Methods List. The only time that a selected control or menu item is affected is if:

- Another class is selected.
- A different property is selected.
- A different property sheet is selected.

Note Selecting a different property in the Properties List causes the Methods List to be reloaded only if the property is a control or menu item event.

Method Status List Browser (NFS 65584)

You can now display the status list of methods and constants that are unsaved or in error in multiple schemas, using the new Method Status List Dialog that is displayed when you select the Browse menu **Status List** command. By default, the current schema is searched but you can search:

- Another schema selected in a combo box.
- All schemas.
- The selected schema and all subschemas.
- The selected schema and all superschemas.

In addition:

- You can specify that the search is performed only for constants and methods that were last changed by you. The provision of this **Last changed by current user only** check box on the Method Status List Dialog has resulted in the removal of the **Status List for Current User** command from the Browse menu.
- The Shift+Ctrl+C shortcut for the Browse Status List for Current User action has changed to the Browse Status List action in the **Browse** shortcut key category on the **Short Cut Keys** sheet of the Preferences dialog.

Painter

This section describes the JADE Painter changes in this release.

Hierarchy for Form Dialog (NFS 65168)

To change the parent of a control from the Hierarchy for Form dialog

1. While holding the Shift key down, left-click the name of the control to be re-parented in the list of control names.
2. While holding down the mouse button, drag the mouse to the new parent control in the Hierarchy for Form dialog and then release the mouse.
3. A message box is then displayed with the following message, prompting you to confirm the parent change.

```
Make 'control-name'  ('control-class-name') a child of 'new-parent-name'
('parent-class-name') and preserve left and top?
```

The message box contains three buttons. Click:

- **Yes**, if you want to change the parent and retain the current top and left position within the new parent.
- **No**, if you want to change the parent and retain the same screen position within the new parent.
- **Cancel**, if you want to abort the parent change process.

Note that while dragging a control, moving the mouse:

- Above the list box entries containing the control and form names scrolls the list up.
- Below the list box entries containing the control and form names scrolls the list down.
- Over the expanded node image causes the list item to be expanded one level.
- Over an item that cannot have control children displays a no-drop allowed style cursor image.
- Over an item that can host the control being dragged as a child displays a hand with pointed finger cursor image, indicating that you can drop the dragged control.

The status text of the completed drag and drop process is displayed at the bottom of the Hierarchy for Form dialog when you release the mouse over an entry in the list box.

Inherited Controls in the Properties Dialog (NFS 65212)

The JADE Painter Properties dialog has been changed as follows, so that you can now select and copy text.

- The value of a property in the right column is displayed with blue text when the selected control is inherited from a superclass, indicating that you cannot change the value.
- When you click a text value in the right column and the selected control is inherited from a superclass, the content of that text box is now read-only instead of disabled.

The behavior of entries that use a combo box for selection does not change, and the combo box is disabled for inherited control values.

Properties and Hierarchy for Form Dialogs (NFS 65238)

If there is another form being painted, the Properties and Hierarchy for Form dialogs now remain open and they are updated when the next painted form is activated.

Selecting a Number of Controls (NFS 46830)

The Controls menu in the JADE Painter now provides the **Select All Children** and **Select All Siblings** commands, which enable you to move a block of controls to another parent, reposition a block of controls, or change properties on a group of controls, for example.

The **Select All Children** command is enabled if the currently selected form or control has children. Clicking this menu item deselects all currently selected controls and selects all immediate children of the current form. If no control is selected, all immediate children of the form or control are selected.

The **Select All Siblings** command is enabled if a control is currently selected and the parent of the control has children other than the currently selected control. Clicking this menu item deselects all currently selected controls and selects all immediate children of the parent control.

In addition, the JADE Painter now provides the following default shortcut keys. (For details about changing default JADE shortcuts, see "Handling Shortcut Keys (NFS 64565)" under "Shortcut and Function Keys", later in this document.)

- Add Control (Ctrl+Insert)
- Delete Control (Delete)
- Select Parent (Ctrl+Shift+P)
- Select All Children Ctrl+Shift+C)
- Select All Siblings (Ctrl+Shift+L)
- Push To Bottom (Ctrl+Shift+B)
- Bring To Top (Ctrl+Shift+T)

- Tab Ordering (Ctrl+Shift+O)
- Wizard (Ctrl+Shift+W)

Separator in the Control Palette (NFS 65251)

The ability to add separators to the **Control** palette has been reimplemented, as follows.

1. The Control Palette Edit dialog now displays a separator image with the caption **Add Separator (Drag)**.
2. The pane at the top of the dialog now displays **To Add Separator   Drag the separator from the image to the control palette**.
3. The **To Remove** description in the pane at the top of the dialog now states **Drag palette button or separator to the waste bucket below**.
4. The **To Reposition** description in the pane at the top of the dialog now states **Drag within palette to move buttons or separators**.
5. The controls already on the **Control** palette that are displayed in the **Controls** list box are now disabled.
6. Any controls not yet added to the **Control** palette that are displayed in the **Controls** list box are enabled and have a background color of yellow.

» **To add a separator to the control palette**

1. Click the separator image on the Control Palette Edit dialog.
2. Drag it to the required position on the **Control** palette.

The process of dragging a **Control** class to the **Control** palette has changed, as follows.

- You can now drag only entries that are not already added to the palette.
- When the control is added to the palette, the entry in the **Controls** list box is disabled and the yellow background is removed.

» **To remove a separator or control icon from the Control palette**

- Drag the palette item to the **Unwanted Buttons** image on the Control Palette Edit dialog.

A control dragged from the **Control** palette in this way is then enabled in the **Controls** list box and displayed with a yellow background.

Any changes that you make to the palette, including adding separators, are stored in your user profile for the selected schema when you click the **Close** button on the Control Palette Edit dialog.

Refactoring Methods

This section describes the refactoring changes in this release.

Declaring a Local Variable when Refactoring (NFS 66264, JEDI 141)

The **Declare Local Variable** command from the Refactor submenu of the Edit menu attempts to automatically determine the type of an unknown variable when it appears on the left- or right-hand part of an assignment statement.

This functionality now also determines the type of a **foreach** variable using the type of expression in the **in** section of the **foreach** instruction. If the **in** expression is:

- A collection, the membership of the collection is used to define the variable type.
- *Not* a collection, the expression type is used as the variable type; for example, **Integer**.

If the type of the expression cannot be determined, the Declare Local Variable Name dialog is displayed, prompting you to specify or select in the combo box the primitive type of the variable.

Refactoring a Method Variable (NFS 65081)

When the cursor is positioned in a method parameter, local variable, or local constant in the editor pane and you select the **Rename / Change** command from the Refactor submenu of the Edit menu, JADE now checks whether the specified new name is already used.

The rename action is rejected if:

- The new name already exists as a local entity such as a parameter, local variable, or constant.
- A property, method, or constant already exists in the selected class or its superclasses.

Refactoring a Method Local Constant (NFS 66261, JEDI 179)

When the cursor is positioned in a string, numeric, or boolean value in a method:

1. Select the **Promote to Method Constant** command from the Refactor submenu of the Edit menu.

   The Set name for Local Method Constant Value dialog is then displayed, showing the selected value.

2. In the **Constant Name** text box, specify the name that you require for the local method constant and then click the **OK** button.

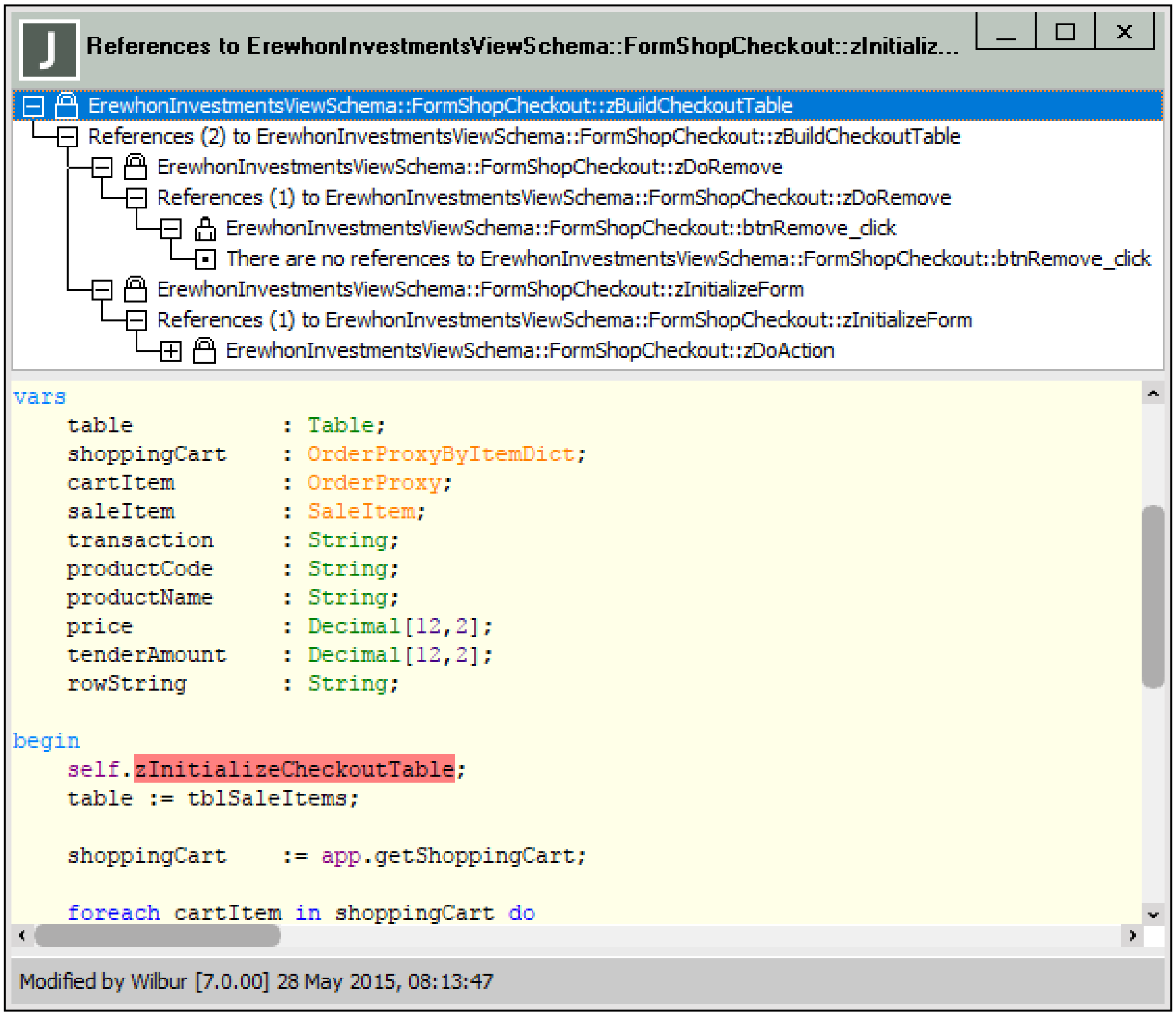
The constant is then added to the **constants** section of the method (which is created if it does not exist) and the selected instance of that constant in your method is replaced by the specified name.

To undo the change or changes, press Ctrl+Z to undo each change in that method.

Note The **Promote to Method Constant** command no longer replaces all instances of that constant, because the value can appear in unrelated contexts; for example, **false**, **0**, and so on.

Reference Results Navigation (NFS 64239)

The References Browser for the display of method references now allows each method reference entry in turn to have its references added to the list, and so on, recursively, as shown in the example in the following image.


References to ErewhonInvestmentsViewSchema::FormShopCheckout::zInitializ...

```
ErewhonInvestmentsViewSchema::FormShopCheckout::zBuildCheckoutTable
  References (2) to ErewhonInvestmentsViewSchema::FormShopCheckout::zBuildCheckoutTable
    ErewhonInvestmentsViewSchema::FormShopCheckout::zDoRemove
      References (1) to ErewhonInvestmentsViewSchema::FormShopCheckout::zDoRemove
        ErewhonInvestmentsViewSchema::FormShopCheckout::btnRemove_click
          There are no references to ErewhonInvestmentsViewSchema::FormShopCheckout::btnRemove_click
    ErewhonInvestmentsViewSchema::FormShopCheckout::zInitializeForm
      References (1) to ErewhonInvestmentsViewSchema::FormShopCheckout::zInitializeForm
        ErewhonInvestmentsViewSchema::FormShopCheckout::zDoAction
```

```
vars
    table           : Table;
    shoppingCart    : OrderProxyByItemDict;
    cartItem        : OrderProxy;
    saleItem        : SaleItem;
    transaction     : String;
    productCode     : String;
    productName     : String;
    price           : Decimal[12,2];
    tenderAmount    : Decimal[12,2];
    rowString       : String;

begin
    self.zInitializeCheckoutTable;
    table := tblSaleItems;

    shoppingCart    := app.getShoppingCart;

    foreach cartItem in shoppingCart do
```

Modified by Wilbur [7.0.00] 28 May 2015, 08:13:47

Each method entry in the references list has an associated **+** (expand) or **-** (collapse) icon. Clicking on the **+** icon for a displayed method in the list box adds references to the method as child entries in the list box. Clicking on the **-** icon again hides the list of references to that method.

Each added method reference can have its references added to the display in the same way, up to a limit of 32 deep.

When the **+** icon is clicked, a child item is added to the list box: This entry is either:

- *There are no references to <method-name>*, when the method has no references.
- *References (<count>) to <method-name>*, when there are references. The reference entries to the method are added as a list box item children of this heading line.

The **Application** class **initialize** and **finalize** methods, property references (condition methods), and RPS mapping references have a leaf icon, and cannot be expanded.

Note The behavior of the Methods menu **References** command for the displayed method is unchanged. It continues to display the references to the selected method in a separate References Browser.

References to initialize and finalize Methods Displayed (NFS 65213)

When performing references on a method and that method is defined to be an **Application** class **initialize** or **finalize** method, the References window now displays an entry listing the name of the schema and application in which the method is referenced; for example:

```
MySchema::MyApplication application initialize method
MySchema::MyApplication application finalize method
```

In addition, if the method is named **initialize** or **finalize** and it is called by default because the application **initialize** or **finalize** method are not specifically set, the application instances are also listed.

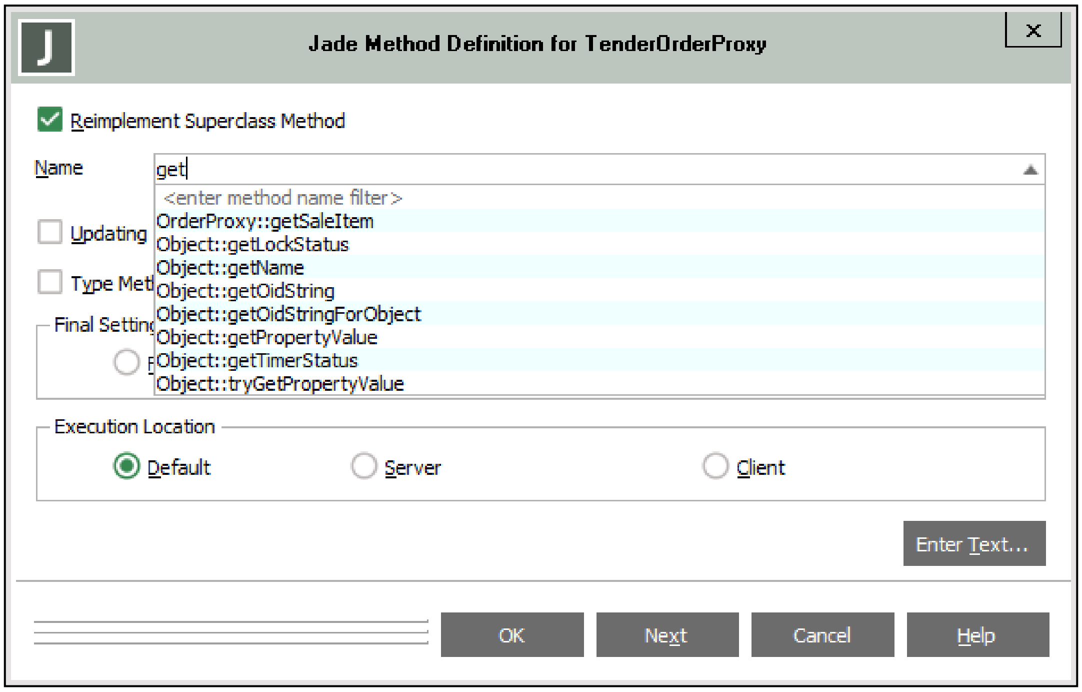
Clicking on the entry in the list box displays only that entry in the editor pane. No other action is available; for example, a context menu command action.

Reimplementing a Superclass Method (NFS 65855)

To improve the user experience when reimplementing a superclass method, the following changes have been made.

- The **Name** combo box displaying the list of superclass method names in the Jade Method Definition dialog has been changed to a **Style_DropDown** (**0**) type so that you can now enter text.
- As the first entry of <enter method name filter> displayed in the combo box drop-down list is a prompt, it is disabled and cannot be selected.
- Initially, all possible superclass methods are displayed as they were in earlier releases.
- When you enter text in the **Name** combo box, only those entries that have the entered text (which is case-insensitive) somewhere in the entries in the drop-down list are displayed. (Those that do not contain the specified text are hidden.)

  As a result, the more text you enter, the shorter the list of possible methods, as shown in the following image.


Jade Method Definition for TenderOrderProxy

[✓] Reimplement Superclass Method

Name: get

<enter method name filter>
OrderProxy::getSaleItem
Object::getLockStatus
Object::getName
Object::getOidString
Object::getOidStringForObject
Object::getPropertyValue
Object::getTimerStatus
Object::tryGetPropertyValue

You can then select the required method by using the mouse or the up and down arrow keys.

See also "Superclass Method Reimplementation (NFS 66343, JEDI 178)", elsewhere in this document.

Removing Properties and Constants from a Class (NFS 66285)

When a property or constant is removed from a class, JADE now retains the same **topIndex** position in the Property List after the deleted entity and selects the next entry in the list (or the prior entry, if it was the last entry in the list). This enables you to delete multiple entries from the list without having to scroll again to locate the next constant or property to be deleted.

Saving Method Source (PAR 65126)

If patch control is enabled and you have unchecked the **Save Source on Every Compile** check box on the **Source Management** sheet of the Preferences dialog to disable the saving of method source each time a method is compiled, you can now press Shift+F8 to save the source during the compilation of a method.

Schema Collection Inspector (PAR 65796)

When the **Show Collection Properties** command in the Options menu of the Schema Collection Inspector form is checked and a collection is displayed, the Object List in the middle pane of the form is now always displayed, showing the:

- Class hierarchy
- Any local properties
- The **edition**, **lastUpdateTranId**, and **creationTime** properties

SDS Primary Interface Deletion (PAR 65710)

Interfaces can now no longer be deleted from the current version of an SDS primary database.

Search Functionality

This section describes the JADE search functionality in this release.

Reverse Direction Search (NFS 66180)

The Edit menu now provides the **Find Again Reverse Direction** command, which performs another search using your last set of search options but in the opposite direction. If the search was specified as a forward search, the reverse search is performed backwards. Conversely, if the search was specified as a backwards search, the reverse search is forwards.

The Shift+F3 shortcut (Find Again Reverse) is listed in the table of **Find** category shortcuts in the **Short Cut Keys** sheet of the Preferences dialog. You can change this shortcut key to a combination of your choice.

Searching by Uppercase Initials (NFS 65559)

The Find Type dialog now enables searching using uppercase initials (for example, entering **JWSC** to find the **JadeWebServiceConsumer** class).

The rules are:

- All text entered in the **Find** text box must be uppercase.
- The first character that you enter must match the first character of a type to be listed in the **Select Required Entry** list box.
- All other entered characters must match the uppercase characters in the name in the list box.
- Any entries that contain the specified text (case-insensitive) are also displayed in the list.

Note The Find Type dialog now lists only entries that match the specified search text.

Searching for an Entity (NFS 64726)

The Find Type dialog, accessed from the F4 shortcut key or the **Find** command from the Schema, Classes, Types, or Interfaces menu, now provides a list of selectable suggestions for what is being entered in the context of the current context.

When you enter text into the **Find** text box, only those entities that contain the specified text somewhere in the type name of the entry are made visible in the **Select Required Entry** list box. For example, if you specify **cust** when searching for a class, only those classes that contain **cust** somewhere in the class name are displayed. (The selection is case-insensitive.)

The first entry that begins with the specified text is selected, as it was in earlier releases. If no entry starts with the specified text, no entry is selected. Use the up and down arrow keys to select and move through the entries in the list.

The **Select Required Entry** list box of the Find Type dialog behaves the same as the **AutoComplete** list box. This behavior also applies when searching for exported package features, global constants, interfaces, schemas, and the partial extract dialog.

Searching for and Extracting Changed Entities (PAR 65688)

The search for checked out methods has been changed to test the modified timestamp of the checked out method or methods against the date range specified in the Checked Out Methods dialog.

When the Checked Out Methods or the Change Identification Browser is displayed, you can then extract changed or checked out methods, by using the common Save As dialog.

Senders Browser Size (NFS 66326, JEDI 98)

The Senders Browser that is used to display the results of various searches (for example, references) is now saved and restored to the previous size and position when the **Save Size and Position** check box on the **Window** sheet of the Preferences dialog is checked (that is, the value is **true**).

In addition, the top panel containing the list box is now reduced if there would be more than one empty line item displayed in the list box. The additional height is assigned to the editor pane instead.

Shortcut and Function Keys

This section describes the JADE shortcut and function key changes in this release. See also "Using the Same Source Window (NFS 65064)", later in this document.

New Class Browser Opened with Ctrl+B (NFS 65313)

When opening a new Class Browser using the Ctrl+B shortcut keys, the class selected in the new browser now defaults to the same class that is currently selected when the action is performed from a hierarchy browser.

Handling Shortcut Keys (NFS 64565)

You can now change most of the development environment and editor shortcut keys to values of your choice; for example, changing the editor Ctrl+S key binding (**Insert Syntax**) to the **Save** function. The exceptions are editor pane accelerator key bindings, standard Windows functionality shortcuts (for example, for copy, paste, and undo actions), and Windows form shortcuts. In addition, you cannot:

- Add a shortcut to a menu item that does not have a shortcut defined in the JADE Painter
- Change the key accelerator in the caption of a menu item; for example, **&File** to **F&ile**

The **Editor Key Bindings** sheet of the Preferences dialog has now been replaced by the **Short Cut Keys** sheet, which enables you to change the defined shortcut keys for the editor key bindings, all of the JADE development environment, RootSchema, and **JadeMonitorSchema** forms.

The table of bindings displayed on the **Short Cut Keys** sheet can be displayed by category; that is:

- Action
- All (the default)
- Browse
- Debugger
- Edit
- Editor
- Find
- Inspect
- Monitor
- Painter
- Unit Test
- Windows

You can sort the table of shortcut keys by shortcut or description, by clicking the respective **ShortCut** or **Description** column heading.

The shortcut key bindings are saved in your user profile and can be extracted to a preferences initialization file by using the existing **Export Preferences** functionality provided on the **Miscellaneous** sheet of the Preferences dialog.

The check boxes on the **Editor Key Bindings** sheet that enabled you to swap the F5 and F9 keys and the F11 and F12 keys have been moved to the **Short Cut Keys** sheet.

The **MenuItem** class **setAccelerator** method that enabled you to dynamically change menu shortcuts at run time, available since JADE 2016.0.01, has been deprecated and replaced by the **MenuItem** class **setShortCutKey** method, which more accurately defines the method behavior. (See also "Window Class Constants", later in this document.)

JADE uses this facility to enable the swap of the F5 and F9 keys and the F11 and F12 keys so that you can change the functionality to match the corresponding keys in Microsoft Visual Studio (that is, F5 for run/continue execution and F9 for set breakpoint, F11 for open source window and F12 for show symbol information).

For more details, see "Maintaining Shortcut Keys", in Chapter 2 of the *JADE Development Environment User's Guide*.

Painter Shortcut and Function Keys (NFS 46901)

The shortcut and function keys listed in the following table are now available from the JADE Painter.

Menu Item	Shortcut or Function Key
New Form	F3
Edit Form	F12
Menu Design	Ctrl+M
Find Control	Ctrl+F
Align Left	Ctrl+L
Align Right	Ctrl+R

Menu Item	Shortcut or Function Key
Align Top	Ctrl+T
Align Bottom	Ctrl+B
Centre Horizontally	Ctrl+Shift+H
Centre Vertically	Ctrl+Shift+V
Same Width	Ctrl+W
Same Height	Ctrl+H
Same Height and Width	Ctrl+Shift+S
Standard Size	Ctrl+S
Grid	Ctrl+G
Edit Control Palette	Ctrl+E
Translate Properties	F6

Painter Shortcut Key Duplication (NFS 65166)

When a shortcut key is selected in the **Shortcut Key** combo box on the **Menu Item** sheet of the Menu Design dialog, the Painter now checks that the shortcut key combination is not already assigned to a menu item on the form being painted, any superclass forms, and any subclass forms.

If the selected shortcut key combination (for example, Ctrl+F2) is already assigned, the *Allow reuse?* warning message is displayed, identifying which schema, form, and menu item already has that shortcut key assigned. You are prompted to click the:

- **No** button if you do not want the shortcut key or keys reused. The **Shortcut Key** combo box is then reset to the previously selected entry.
- **Yes** button, to update the menu item with the selected shortcut key or keys.

Source Control

You can now use the built-in Git client source control feature to work on files already extracted from or loaded into the Git client working directory. The cloning, staging and committing, pushing, pulling, and checking out operations are accessed from JADE development environment menu commands. The JADE Git client functionality is fully compatible with other Git clients that you can use for more-specialized operations.

Note The JADE product information library documents only the JADE implementation of the Git source control; it does not cover the Git technology itself. (If you are not familiar with Git, see https://git-scm.com/book/en/v2.)

JADE incorporates a Git client, so that you can use an on-premises or a cloud-based Git provider to move your source and source changes into (push) and out of (pull) your team's source repository, to more-easily incorporate your JADE source into a modern development workflow.

Note The Git client is supported on both presentation and standard clients.

The Source Control feature comprises mainly an integrated Git client working on files extracted to or loaded from the Git working directory, using the JADE development environment Browse menu **Git Source Control Client** command and its submenu commands.

The JADE development environment provides source control security hooks. Access to the Git client requires authentication and authorization.

See also:

- "Source Control Functions", in Chapter 2 of the *JADE Object Manager Guide*, for details about source control development environment tasks to which you can control access by using the **jadeDevelopmentFunctionSelected** function **taskName** and **entityName** input parameters.
- "Source Control Authentication", in Chapter 2 of the *JADE Development Environment User's Guide*, for details about using Git Credential Manager for Windows (GCM) to provide authentication and secure Git credential storage.
- "Development Source Control", in Chapter 2 of the *JADE Development Environment User's Guide*, for details about the **Git Source Control Client** submenu commands that enable you to configure your source control client, clone the remote repository, stage and commit source changes, and to push to and pull from the remote repository.

String Browser Case-sensitivity (NFS 64617)

The String Browser now provides the **Case Sensitive** check box at the upper right of the **Search** combo box. This check box, which controls whether a search for a translatable string is case-sensitive, is checked by default. This check box affects a standard prefix or a wildcard search.

Notes For a standard prefix search, a non-case-sensitive search is not as efficient as a case-sensitive search because the existing (case-sensitive) translatable string dictionary must be searched in a linear fashion from the starting character prefix to locate matching strings. As a wildcard search must search the entire dictionary, efficiency is not affected.

The translatable list box continues to display the translatable strings in case-sensitive order, regardless of the setting of the check box.

Summary of Patches Form (NFS 64230)

The Summary of Patches form now uses the background color, foreground color specified in **User Objects**, and font values specified on the **Window** sheet of the Preferences dialog to display the patches summary table.

In addition, the table now automatically sizes the table entries and displays a horizontal scroll bar, if required.

Superclass Method Reimplementation (NFS 66343, JEDI 178)

When a Methods List is displayed in the JADE development environment, the displayed method name now indicates the methods that are reimplementations of superclass methods, by including **(r)** on the end of the list item method name; for example:

```
displayCustomer (r)
```

Note This is not done, however, for event methods, which are always reimplementations of a superclass method.

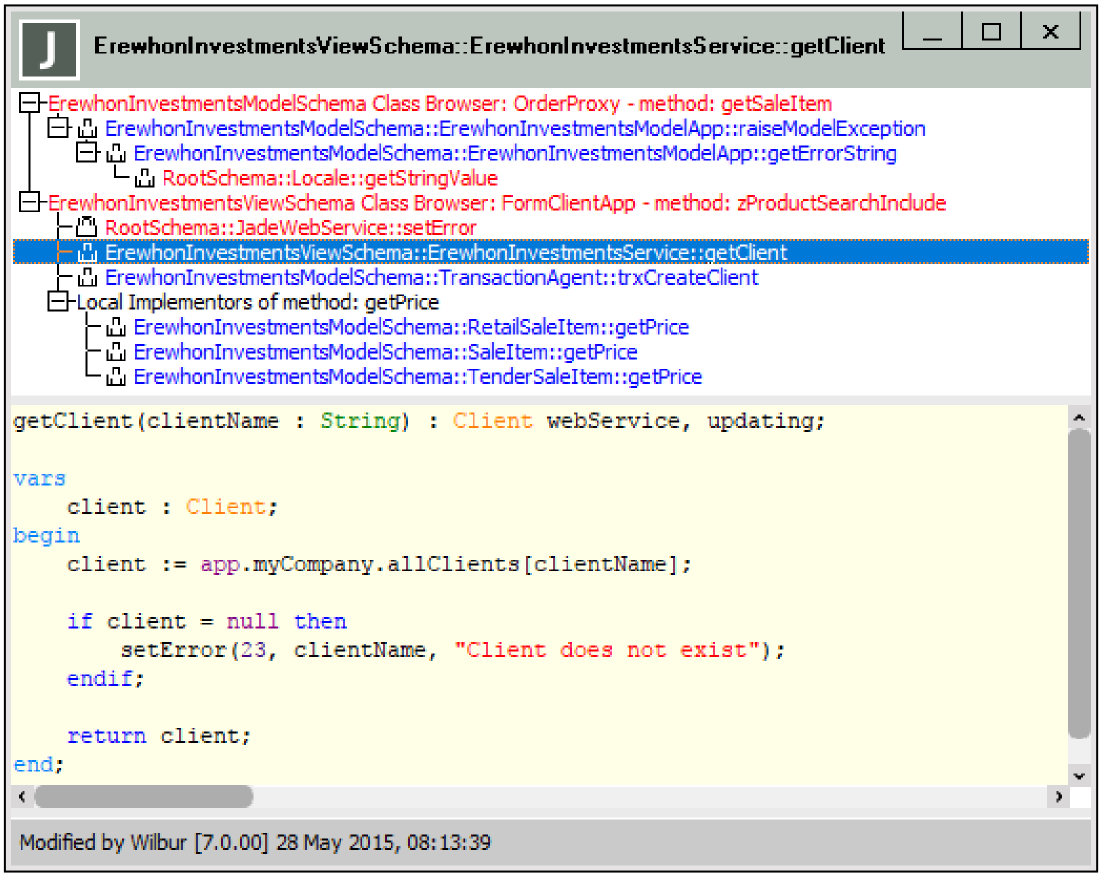
When a reimplemented superclass method is displayed in the editor pane, **(reimplemented)** is also displayed in the status bar of the browser.

Using the Same Source Window (NFS 65064)

The F11 shortcut key functionality has been enhanced, as follows.

- When the **Reuse Same Method Source Window For All** check box on the **Source Management** sheet of the Preferences dialog is checked, pressing F11 on a method name in the editor pane logic now adds the list of local implementors of that method to the methods list in the History List of the Reused Method Source form instead of displaying a separate window to display the list of implementors.

A parent entry of those methods called **Local Implementors of Method: *method-name*** is added to that list box.



```
ErewhonInvestmentsModelSchema Class Browser: OrderProxy - method: getSaleItem
    ErewhonInvestmentsModelSchema::ErewhonInvestmentsModelApp::raiseModelException
        ErewhonInvestmentsModelSchema::ErewhonInvestmentsModelApp::getErrorString
            RootSchema::Locale::getStringValue
ErewhonInvestmentsViewSchema Class Browser: FormClientApp - method: zProductSearchInclude
    RootSchema::JadeWebService::setError
    ErewhonInvestmentsViewSchema::ErewhonInvestmentsService::getClient
    ErewhonInvestmentsModelSchema::TransactionAgent::trxCreateClient
    Local Implementors of method: getPrice
        ErewhonInvestmentsModelSchema::RetailSaleItem::getPrice
        ErewhonInvestmentsModelSchema::SaleItem::getPrice
        ErewhonInvestmentsModelSchema::TenderSaleItem::getPrice
```

```
getClient(clientName : String) : Client webService, updating;

vars
    client : Client;
begin
    client := app.myCompany.allClients[clientName];

    if client = null then
        setError(23, clientName, "Client does not exist");
    endif;

    return client;
end;
```

Modified by Wilbur [7.0.00] 28 May 2015, 08:13:39

You can toggle the display of the methods list by clicking in the **+** (expand, or show) icon or the **-** (collapse, or hide) icon of the parent entry.

Clicking on the parent entry displays only the list box item text in the editor pane.

- The **Source Management** sheet of the Preferences dialog now provides the **Reuse Same Method Source Window For Each Origin** check box and the existing **Reuse Same Method Source Window** check box has now been renamed **Reuse Same Method Source Window For All**.

  Only one of these check boxes can be checked. Checking one check box unchecks the other.

  Checking the **Reuse Same Method Source Window For All** check box results in the same behavior as that of the **Reuse Same Method Source Window** check box in earlier releases; that is, it opens a new source window and adds entries into the same single copy of the extended source window, listing methods for all used browser windows.

  Checking the new **Reuse Same Method Source Window For Each Origin** check box results in one History List of the Reused Method Source form for each browser origin; for example, pressing F11 on a method name in a method in the editor pane opens an extended source window that is populated only with actions in the original browser or in the displayed extended source window (the History List of the Reused Method Source form). Performing actions that cause use of a source window in another browser opens a new extended source window rather than using any other open source window not initiated by that browser.

  These values are saved in your user profile when you extract your user preferences to a file and they are restored when you reload your preferences.

The View menu from a Hierarchy Browser now also provides the **Use Same Method Source Window For All** and **Use Same Method Source Window For Each Origin** commands that toggle these values, with a check mark at the left of a command indicating the current value. These options are set from your user profile settings when the form is opened or if your user preference is changed.

You can check or uncheck these options for each open Hierarchy Browser. (Checking one command unchecks the other.)

Note If the **Reuse Same Method Source Window For Each Origin** value is checked and the method about to be displayed is open in another window and has been modified, the **Reuse Same Method Source Window For Each Origin** option is ignored and that window will be opened instead.

Clicking the *form* entry in the methods list in the History List of the Reused Method Source form returns you to the originating form. (As the original method can be modified, displaying another copy of the method source is not desirable.)

JadeTableSheet Class

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

Column Content Filling Total Width (NFS 65206)

The **JadeTableSheet** class now provides the **extendedColumn** integer runtime-only property, which defaults to zero (**0**).

This property specifies that when the **Table** class **autoSize** property is set to **true**, after the table columns are auto-sized to fit the minimum size for their contents and the total width of the visible columns is less than the value of the **clientWidth** property, the specified column is enlarged to use the remaining space. The value of the **JadeTableColumn** class **maxColumnWidth** property is ignored.

The value of the **extendedColumn** property is ignored unless all of the following are **true**.

1. The **autoSize** property is set to **AutoSize_BothColumnMinimum** or **AutoSize_ColumnMinimum**.
2. The **extendedColumn** property is set to a valid visible column number.
3. The **columnWidth** property has not been set manually by the user or by logic.
4. The total width of all visible columns is less than the value of the **clientWidth** property of the table.

Pixel-based Scrolling in Tables (NFS 64888)

The **JadeTableSheet** class now provides the **pixelHorzScrollIncrement** and **pixelVertScrollIncrement** properties, which have an **Integer** value. The default value of both properties is **1**, and the value can be in the range **1** through **32767**. Values outside of this range are treated as **1**.

You can set these properties at run time on a table sheet, to control the number of pixels that are scrolled when the scrolling mode of the sheet is pixels (that is, the value of the **JadeTableSheet** class **scrollMode** property is **ScrollMode_HorzPixel_VertCell** (1), **ScrollMode_VertPixel_HorzCell** (2), or **ScrollMode_Both_Pixel** (3).

The **pixelHorzScrollIncrement** and **pixelVertScrollIncrement** properties enable the amount of scrolling to be increased. For example, by setting the vertical pixel increment to the height of the displayed text in the table, the scrolling would scroll a line at a time.

The increment value is used only when the user clicks on a scroll bar arrow or scrolls using the mouse wheel and the scrolling mode is pixels for that scroll bar.

Note When scrolling with the mouse wheel, the scrolling amount is multiplied by an increment set by the user. (This increment is usually **3**.)

The **pixelHorzScrollIncrement** property value is ignored unless the value of the **scrollMode** property of the sheet is **ScrollMode_Both_Pixel** or **ScrollMode_HorzPixel_VertCell**.

The **pixelVertScrollIncrement** value is ignored unless the value of the **scrollMode** property of the sheet is **ScrollMode_Both_Pixel** or **ScrollMode_VertPixel_HorzCell**.

The following code fragment is an example of the use of the **pixelVertScrollIncrement** property.

```
table1.accessSheet(1).pixelVertScrollIncrement := 14;
```

Loading Schemas using an Application (NFS 64795, 65133)

JADE has now implemented a new RootSchema application called **JadeSchemaLoader**, which is a non-GUI application that can be run:

- As a presentation (thin) client

  The loader parameters, or arguments, are specified between the **startAppParameters** and the optional **endAppParameters** arguments. (When the **startAppParameters** argument is encountered, all subsequent parameters up to the optional **endAppParameters** argument or the end of the command line are collected into a **HugeStringArray** and passed to the **initialize** method of the application.)

  ```
  jade.exe schema=RootSchema app=JadeSchemaLoader
  ini=JADE-initialization-file-path
  AppServer=remote-TCP/IP-address-or-name-of-application-server
  AppServerPort=TCP/IP-port-number-of-application-server startAppParameters
  load-command-line-arguments endAppParameters
  ```

  The following is an example of the command line for loading a **Test** schema from a presentation client.

  ```
  jade.exe schema=RootSchema app=JadeSchemaLoader path=d:\jade\system
  ini=d:\jade\myjade.ini AppServer=MyAppServer AppServerPort=1234
  startAppParameters schemaFile=d:\jade\scm\test.scm ddbFile=d:\jade\scm\test.ddb
  dontSaveSources=true deletePropertiesIfAbsent=true
  loadStyle=onlyStructuralVersioning replayableReorg=true
  compileUnchangedMethods=true endAppParameters
  ```

- From **jade.exe** in single user or multiuser mode as a standard (fat) client

  The loader parameters are specified after the **startAppParameters** argument.

  ```
  jade.exe schema=RootSchema app=JadeSchemaLoader
  ini=JADE-initialization-file-path path=database-path
  server=multiUser|singleUser startAppParameters load-command-line-arguments
  ```

  The following is an example of the command line for loading a **Test** schema from a standard (fat) client.

  ```
  jade.exe schema=RootSchema app=JadeSchemaLoader path=d:\jade\system
  ini=d:\jade\myjade.ini startAppParameters schemaFile=d:\jade\scm\test.scm
  ddbFile=d:\jade\scm\test.ddb dontSaveSources=true deletePropertiesIfAbsent=true
  loadStyle=onlyStructuralVersioning replayableReorg=true
  compileUnchangedMethods=true
  ```

- From **jadclient.exe** in single user or multiuser mode from a non-GUI client

  The loader parameters are specified after the **startAppParameters** argument.

  ```
  jadclient.exe schema=RootSchema app=JadeSchemaLoader
  ini=JADE-initialization-file-path path=database-path
  server=multiUser|singleUser startAppParameters load-command-line-arguments
  ```

The following is an example of the command line for loading a **Test** schema from a non-GUI client.

```
jadclient.exe schema=RootSchema app=JadeSchemaLoader path=d:\jade\system
ini=d:\jade\myjade.ini startAppParameters schemaFile=d:\jade\scm\test.scm
ddbFile=d:\jade\scm\test.ddb dontSaveSources=true deletePropertiesIfAbsent=true
loadStyle=onlyStructuralVersioning replayableReorg=true
compileUnchangedMethods=true
```

- From your application code, by calling the **Application** class **startApplicationWithParameter** method, passing the command line arguments in a shared transient HugeStringArray; that is:

```
app.startApplicationWithParameter("RootSchema", "JadeSchemaLoader",
parameters);
```

The result of the load action is returned as the last parameter in the array. A zero (**0**) value indicates no error.

```
"result=error-code"
```

The following is an example of the JADE code for loading the **Test** schema.

```
vars
    parameters : HugeStringArray;
begin
    beginTransientTransaction;
    create parameters sharedTransient;
    parameters.add("schemaFile=d:\jade\scm\test.scm");
    parameters.add("ddbFile=d:\jade\scm\test.ddb");
    parameters.add("dontSaveSources=true");
    parameters.add("deletePropertiesIfAbsent=true");
    parameters.add("loadStyle=onlyStructuralVersioning");
    parameters.add("replayableReorg=true");
    parameters.add("compileUnchangedMethods=true");
    commitTransientTransaction;
    app.startApplicationWithParameter(rootSchema.name, "JadeSchemaLoader",
        parameters);
end;
```

All command line arguments supported by **jadloadb** are available, including all schema load options, loading JADE Report Writer files, initiating reorganizations, and executing methods. Each phase of the load operation and error messages are output to the **JadeSchemaLoader.log** file. A load failure is indicated by a non-zero exit code from the process.

This feature enables you to deploy schema changes directly to an application server running in single user mode without having to stop any applications that are running. An online database reorganization can be performed and applications need to be shut down only for the transition.

Caution For details about **JadeSchemaLoader** application security, see "JadeSchemaLoader Application Security Considerations", in the following subsection.

JadeSchemaLoader Application Security Considerations

The **JadeSchemaLoader** application poses a potential security risk to JADE databases.

If a JADE environment includes a web-facing application server, the **JadeSchemaLoader** application can be run by any presentation (thin) client connected to the internet. Such a client could be used to load an arbitrary schema that could include a malicious JADE application that the user could subsequently run to gain access to the JADE database.

You should enable application restrictions to prevent the unauthorized running of the **JadeSchemaLoader** application. For details, see "Controlling the JADE Thin Client Application Execution", in Chapter 3 of the *JADE Thin Client Guide*; for example:

```
[JadeAppServer]
EnableAppRestrictions=true
AllowSchemaAndApp1=MySchema,MyApp
AllowSchemaAndApp2=MySchema,MyOtherApp
```

In this example, no RootSchema applications can be run using a presentation client. Only the applications specified in the JADE initialization file **AllowSchemaAndApp<*n*>** parameters in the specified **MySchema** schema can be run by any presentation clients connecting to all application servers.

Lock Timeouts Global Constant

The **LockTimeouts** category now provides the **LockTimeout_Process_Defined** global constant that has the Integer value of **-2** and specifies that the process-defined default lock timeout is used.

This can be used in association with the **Process** class **setDefaultLockTimeout** method.

ODBC OID Field Separator Customization (PAR 65730)

You can now customize the OID field separator, by specifying a single punctuation or similar character in the **OidFieldSeparator** parameter in the [JadeOdbc] section of the JADE initialization file on the server; for example:

```
[JadeOdbc]
OidFieldSeparator=@
```

The JADE ODBC driver formats and parses OID strings using the specified separator.

Customizing the OID field separator can be useful when the default OID String values that are produced are misinterpreted by a third-party tool as a different ODBC type such as a Decimal.

One character only can be specified. You can specify only a punctuation or similar character; alphanumeric (alphabetic and numeric) characters and whitespace are not allowed. When the value of this parameter is set to an invalid character or a whitespace, a message is logged and the default value of a period (**.**) is used.

When using the JADE ODBC thin client driver, this parameter must be set in the JADE initialization file for the server application.

Note The **OidFieldSeparator** parameter is global; that is, it takes effect for every ODBC query made in your JADE system.

This parameter is read when the ODBC connection is established.

Printing

This section describes the printing changes in this release.

Default Printer (NFS 66216)

When JADE printing opens the default printer in earlier releases, it used the older-style Microsoft facility to obtain the default printer, which involved calling the API, as follows.

```
int len = GetProfileString("windows", "device", NULL, szPrinterName, sizeof
(szPrinterName));
```

Windows uses this to look up the registry settings and returns the default printer name, driver, and port. However, this causes problems under Citrix when the registry settings have not been set up.

JADE now uses the Microsoft **GetDefaultPrinter** API to find the default printer name. If this API fails, JADE attempts to find the default printer using the original older-style API.

Printer Class getAllPrinters Method (NFS 64866)

In earlier releases, an exception was raised if you called **app.printer.getAllPrinters** and one or more printers had a very long name (for example, if information was redirected to a terminal server session).

To allow for very long printer names, the **StringArray** parameter of the **Printer** class **getAllPrinters** method is now an **Array**; that is the signature of the **Printer** class **getAllPrinters** method is now:

```
getAllPrinters(sa: Array input): Integer updating;
```

The array type passed to the method must be an array with a membership of **String**, to allow the array type to be a **HugeStringArray** or a **StringArray**. If another array type with a membership other than **String** is passed to the method, exception 1000 *(Invalid parameter type)* is raised.

Note As this change does not affect your existing logic, it does not need to be re-compiled.

Processes

This section describes the process changes in this release.

Getting and Sending Method Cache Statistics (NFS 64890)

The **Process** class now provides the following methods that enable you to get and send the method cache statistics of the executing process. (For details about displaying method cache statistics in the JADE Monitor, see "Method Cache Statistics in the JADE Monitor (NFS 64890)", later in this document.)

- The **getMethodCacheStatistics** method, which has the following signature, retrieves information about the method cache of the executing process and stores it in the passed in **JadeDynamicObject**.

  ```
  getMethodCacheStatistics(jdo: JadeDynamicObject input);
  ```

  The **JadeDynamicObject** is updated to include properties that contain statistics about the method cache and string pool of the executing process.

- The **sendMethodCacheStatistics** method, which has the following signature, programmatically requests a target process (the receiver) to send a notification containing statistics about the method cache of the target process.

  ```
  sendMethodCacheStatistics() serverExecution;
  ```

  The target process can be any current process, including the requesting process itself or a process executing on another node.

  To retrieve the call stack information and send the notifications, the target process is activated temporarily or interrupted, after which it resumes whatever it was doing.

  Note The **sendMethodCacheStatistics** method is asynchronous; that is, it does not wait until the information is received. The information is received through notifications sometime after the method is called.

The **JadeProcessEvents** category now provides the **Process_Method_Cache_Stats_Event** global constant.

Getting and Setting Process Limits (NFS 12719)

The **Process** class now provides the following methods that enable you to get and set the method cache limit and string pool limit of the executing process.

- The **getMethodCacheLimit** method, which has the following signature, retrieves the method cache limit for the executing process.

  ```
  getMethodCacheLimit(): Integer64 clientExecution;
  ```

- The **setMethodCacheLimit** method, which has the following signature, programmatically sets the method cache limit for the executing process.

  ```
  setMethodCacheLimit(limit: Integer64) clientExecution;
  ```

  The **limit** parameter represents the number of bytes to which to set the method cache limit.

  The **limit** value must be greater than the minimum allowed cache size (64K bytes); otherwise exception 1002 (*Invalid parameter value*) is raised.

- The **getStringPoolLimit** method, which has the following signature, retrieves the string pool limit for the executing process.

  ```
  getStringPoolLimit(): Integer64 clientExecution;
  ```

- The **setStringPoolLimit** method, which has the following signature, programmatically sets the string pool limit for the executing process.

  ```
  setStringPoolLimit(limit: Integer64) clientExecution;.
  ```

  The **limit** parameter represents the number of bytes to which to set the string pool limit.

  The **limit** value must be greater than the minimum allowed string pool size (64K bytes); otherwise exception 1002 (*Invalid parameter value*) is raised.

All four methods can be called only on the process instance of the current process. Exception 1265 (*Environmental object operation is out of scope for process*) is raised if you call one of these methods on an instance for another process.

Method Cache Default Value and Lifetime (NFS 64892)In this release:

- The method cache now includes a method lifetime, which represents the retention time before a method can be removed from cache. When a method is executed, the JADE interpreter must load the method code into the interpreter method cache for execution.

  This new functionality works in conjunction with the behavior in earlier releases, so that an old unused method is discarded only if it is older than the specified method lifetime. Although this means that more method cache overruns may occur, methods should be reloaded less frequently, limiting excessive CPU consumption.

- The [JadeInterpreter] section of the JADE initialization file now contains the **MethodCacheLifetime** parameter, which specifies the number of milliseconds a method must be in cache before it is made available for removal.

  This parameter takes an Integer value and must be set to zero (**0**) or greater. Zero (**0**) represents no lifetime, indicating that there is no change from the behavior of earlier releases.

  The default value is zero (**0**) for 32-bit nodes; **600000** (that is, 10 minutes) for 64-bit nodes.

- The default method cache, defined in the default **MethodCacheLimit** parameter in the [JadeInterpreter] section of the JADE initialization file, has been increased from **512K** to **5M** for 64-bit nodes. The default value for 32-bit nodes remains **512K**.

  This also affects the value of the **StringPoolLimit** parameter, which in earlier releases determined its default value as the greater of **5M** or the method cache limit. The default value for this parameter is now also **5M**.

This new behavior works in conjunction with the behavior in earlier releases, so that when the method cache is full, the oldest unused methods are discarded until there is enough room for the new method or the oldest method is younger than the specified method lifetime.

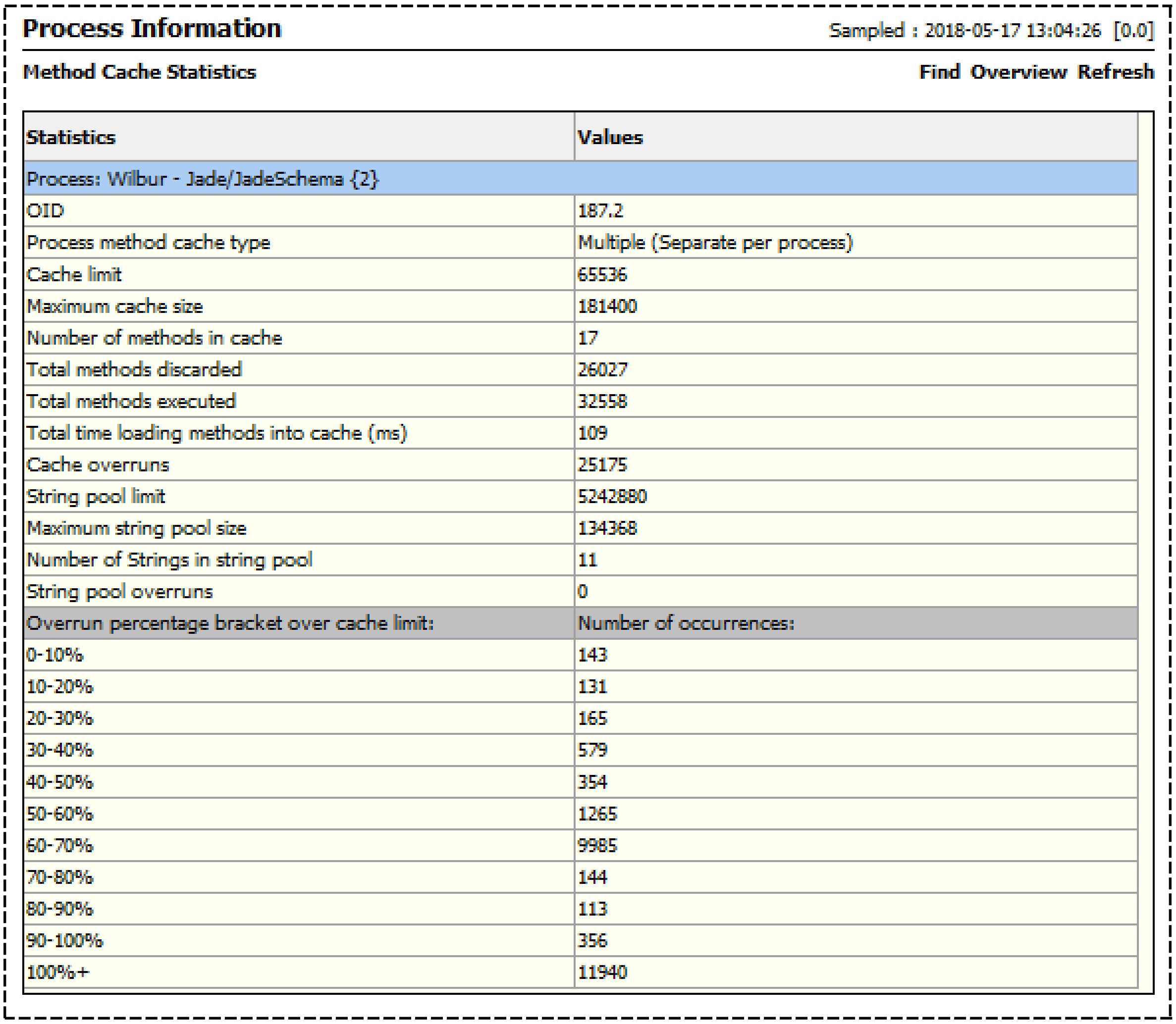
Method Cache Statistics in the JADE Monitor (NFS 64890)

The JADE Monitor now enables you to display method cache statistics for a selected process or for all processes in a node, as follows.

1. Select the **Users** activity in the in the Navigator pane. The **Users** view is then displayed.
2. Right-click on the node in the first row of the table, or on a process in the node.
3. Select the **Method Cache Statistics** command from the popup menu that is displayed.

The **Method Cache Statistics** table is then displayed in the **Process Information** view.

If the method cache is overrun (for example, you have set it to a low limit such as 64K in the [JadeInterpreter] section of the JADE initialization file), an additional **Overrun percentage bracket over cache limit** table is displayed under the cache statistics for each process, as shown in the following image.



Process Information

Sampled : 2018-05-17 13:04:26 [0.0]

Method Cache Statistics

Statistics	Values
Process: Wilbur - Jade/JadeSchema {2}	
OID	187.2
Process method cache type	Multiple (Separate per process)
Cache limit	65536
Maximum cache size	181400
Number of methods in cache	17
Total methods discarded	26027
Total methods executed	32558
Total time loading methods into cache (ms)	109
Cache overruns	25175
String pool limit	5242880
Maximum string pool size	134368
Number of Strings in string pool	11
String pool overruns	0
Overrun percentage bracket over cache limit:	Number of occurrences:
0-10%	143
10-20%	131
20-30%	165
30-40%	579
40-50%	354
50-60%	1265
60-70%	9985
70-80%	144
80-90%	113
90-100%	356
100%+	11940

See also "Getting and Sending Method Cache Statistics (NFS 64890)", earlier in this document.

Progress Displayed on the Taskbar Application Icon (NFS 65259)

JADE now shows the state and progress of an application task in its icon in the Windows taskbar.

If the **Form** class **setTaskBarState** or **setTaskBarProgress** method is called:

- On an MDI child form, the value is applied to the MDI frame form, because the MDI child form does not have an icon on the task bar.
- Before the form is shown (for example, in the **load** method), the value is applied when the form is shown, because there is no taskbar icon before the form is made visible.

Notes This functionality is available only if the application displays an icon on the Windows taskbar. It does not apply to icons in the system tray.

The method calls can silently fail for some earlier versions of Windows (for example, Windows 8.0), even though the Microsoft documentation states that the functionality is available from Windows 7 and later.

The following class constants, methods, and property have been implemented.

Form Class setTaskBarState Method

The **Form** class **setTaskBarState** method, which has the following signature, sets the state of the taskbar icon for the application.

```
    setTaskBarState(state: Integer);
```

The **state** parameter of this method can contain one of the following new constants in the **Form** class.

Form Class Constant	Value	Description
TaskBar_State_NoProgress	0	Hides the progress state on the icon
TaskBar_State_Indeterminate	1	Causes a continuous icon progress state, drawing the state in green from 0 through 100 percent, repeated, indicating the action is in progress but the progress completion time is unknown
TaskBar_State_Normal	2	Displays the current progress state in green
TaskBar_State_Error	4	Sets the progress state to be drawn in red to indicate an error state
TaskBar_State_Paused	8	Displays the current progress state in yellow, to indicate that the action has been paused

Any other value passed to the **setTaskBarState** method is treated as **TaskBar_State_Normal** (2).

Form Class setTaskBarProgress Method

The **Form** class **setTaskBarProgress** method, which has the following signature, sets the extent of the progress to be displayed on the icon.

```
    setTaskBarProgress(value: Integer; maxVal: Integer);
```

Calling this method causes the progress indicator to be displayed in its set state.

If the current state was **TaskBar_State_NoProgress** or **TaskBar_State_Indeterminate**, the state becomes **TaskBar_State_Normal**.

The **setTaskBarProgress** method **value** parameter specifies the current progress value and the **maxVal** parameter specifies the maximum value that will be reached. Both values are relative to zero (**0**).

Note If two forms call the **setTaskBarProgress** method while the taskbar progress is displayed, the progress displays the lowest value.

To hide the progress state when the action is completed, call the **setTaskBarState** method with the **state** parameter set to **TaskBar_State_NoProgress**.

ProgressBar Class showTaskBarProgress Property

The **ProgressBar** class **showTaskBarProgress** property, which has a **Boolean** value, can be set in the JADE Painter and at run time to control whether the progress bar state is shown on the taskbar icon of the application as well as in the progress bar.

The default value of **false** indicates that no progress is displayed on the taskbar icon.

Setting the **showTaskBarProgress** property value to **true** displays progress on the taskbar icon. The Form class **setTaskBarProgress** method is automatically called when the value of the **partsDone** property is updated.

When the progress bar value reaches 100 percent, the **Form** class **setTaskBarState(TaskBar_State_ NoProgress);** method is automatically called to hide the taskbar state display.

If the value of the property never reaches 100 percent, it is your responsibility to ensure that the taskbar progress state is hidden, if required.

The taskbar state is hidden when the form is unloaded.

REST Services

This section describes the REST Service changes in the JADE 2018.0.01 release.

REST Service Call of Protected or Read-only Properties (NFS 64481)

The handling of REST services has now changed so that the application definition includes whether protected properties are excluded from serialization and object construction. This flag is controlled by the **Exclude Protected Properties** check box on the **Web Options** sheet of the Define Application dialog. When this check box is:

- Unchecked (the default), protected properties are included in any serialization of the object and will be included when constructing the object from the received XML or JSON structure.
- Checked, protected properties are not included in any serialization of the object and are ignored when constructing the object from the received XML or JSON data.

The setting of this check box is loaded when the REST service is initiated, and any change to the check box setting does not take effect until the REST service application is restarted.

Note Read-only properties are always excluded from object construction from the received XML or JSON data.

If access of an input object to a REST service call is protected or read-only, REST does not set the property.

REST Service Data Format (PAR 65414)

This handling of data has now been changed so that data received for read-only properties is ignored for both JSON and XML formats.

REST Service Request Exceptions (NFS 66280, JEDI 100)

The response of a REST call is now formatted in the style specified in the URL. When the output format is:

- JSON (Microsoft JSON), the response is formatted as follows.

  ```
  {"__type":"Fault",
  "errorCode":"<error number>",
  "errorItem":"<Exception type>",
  "errorText":"<exception description>"}
  ```

- JSONN (NewtonSoft JSON), the response is formatted as follows.

```
{"$type":"Fault",
"errorCode":"<error number>",
"errorItem":"<Exception type>",
"errorText":"<exception description>"}
```

REST Services Shared Transient Responses (PAR 65735)

A returned shared transient object is now no longer deleted on completion of the REST Services processing. It would be unsafe to do so, because JADE cannot be certain of whether that is the intention and JADE would have to go into transaction state to do so.

Note Anything added to the **JadeRestService** class **objectsToBeDeleted** collection is expected be to non-shared transients, and any other object lifetime will cause the logic to fail because the logic is not in transient state.

Run Application Dialog (NFS 65904)

You can now specify parameters for command line arguments when running an application from within JADE if the initialize method for the application has a single parameter of type **HugeStringArray**.

The **Parameters** text box on the Run Application dialog is enabled when the initialize method for the application has a single parameter of type **HugeStringArray**.

A shared transient **HugeStringArray** object instance is created and populated with the specified parameters when the application runs. Each entry in the array is a space-delimited token from the parameters string; for example:

```
"schema=RootSchema app=JadeSchemaLoader ini=MyJade.ini
schemaFile=d:\jade\scm\test.scm dontSaveSources=true
loadStyle=onlyStructuralVersioning"
```

Note It is up to the application being initiated to delete the shared transient **HugeStringArray** instance.

SDS Secondary Commit-Consistent Backup (NFS 66069, JEDI 160)

JADE now provides an SDS secondary commit-consistent (coherent) backup mechanism, which enables you to create a backup of a secondary that when recovered, stops tracking and is in a commit-consistent (coherent) state.

Error 3221

A new error message 3221 (*SDS primary not connected*) occurs when an operation is initiated that cannot proceed because the secondary is not connected to the primary. If this exception is raised, establish the connection between the secondary and the primary.

JadeDatabaseAdmin Class commitCoherentBackup Method

The **JadeDatabaseAdmin** class now provides the **commitCoherentBackup** method, which has the following signature.

```
commitCoherentBackup();
```

This method signals the successful completion of a database backup transaction on an SDS secondary. It arranges a database quiet point on the primary to establish commit-consistent end backup recovery state that is stored in the database control file, which is then backed up to the default backup directory specified in the **beginBackup** method. Backup recovery stops tracking at end backup, saving replay state.

The commit-consistent checkpoint processing on the primary performs a checkpoint within a database quiet point. The primary **MaxWaitForQuietPoint** value specified in the [PersistentDb] section of the JADE initialization file is used when waiting for the quiet point.

You may encounter an exception 3077 (*Maximum time to wait for a quiet point was exceeded*), in which case, the backup remains active and you can retry the **commitCoherentBackup** method call or you can call the **abortBackup** method. The backup terminates with the exception if it is unhandled. Other errors (for example, the 3221 (*SDS primary not connected*) or 3212 (*SDS a response was not received within a reasonable timeframe*) error) terminate the backup with that exception.

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.)

Button and JadeMask Control Transparency (NFS 65819, JEDI 115)

The handling of the **Button** and **JadeMask** skins has been changed so that when a **Button** or **JadeMask** control is skinned and at least one of the skin images is 32-bit (which supports transparency), the control is now treated as though it is transparent; that is, the control is painted on its parent without the area being erased with the effective value of the **backColor** property. Instead, the parent shows through any transparent areas of the images (for rounded corners, for example). In addition, any semi-transparent parts of the images are anti-aliased with the parent image.

Check Box Skins on Table Controls (NFS 65598, JEDI 45)

The **JadeSkinTable** class now provides the **myCheckBoxSkin** property, which has the type **JadeSkinCheckBox** and availability of read or write at any time. This enables you to define a skin for a **Table** control and specify a check box skin that is used when drawing a cell that has the **inputType** property set to **InputType_CheckBox** or a cell control set to a **CheckBox** control.

Set the value for a check box skin in the **Table** control types on the **Controls** sheet of the Jade Skin Maintenance dialog.

If you want a check box skin to be used when drawing a cell that has the **inputType** property set to **InputType_ CheckBox** or a cell control set to a check box control, select the check box skin that you required from the list of all available check box skins displayed in the list portion of the **CheckBox Skin** combo box. The default value of **<none>** indicates that check boxes in table cells are drawn without a skin.

Color of Selected Items in List Box and Table Skins (NFS 65599)

The **JadeSkinListBox** and **JadeSkinTable** classes now provide the:

- **selectionColorText** property, which controls the text color of a selected item in a skinned list box or table. The default value of **#80000000** (that is, transparent) means that the default selection text color defined by Windows is used.
- **selectionColor** property, which controls the color that is used to draw the background color of a selected item in a skinned list box or table. The default value of **#80000000** means that the default selection background color defined by Windows is used.

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 (NFS 65299)

In JADE 2016, documentation for PAR 65263 stated that if the menu skin associated with a form's skin had only default values set, the menu was not drawn using the menu line properties on the form skin but was drawn in the default non-skinned manner. This restriction has been now been removed.

Skins for menus are now handled as follows.

- The **Menu line height** text box on the **Menu** sheet of the Jade Skin Maintenance dialog has been renamed the **Menu item height** text box. The default value is **0** pixels. (The menu *line* is the menu line visible on the form. The menu *item* is the item, or command, in the popup menu.)
- The **JadeSkinForm** class now provides the Boolean **useMenuLineSkinForMenus** property, which controls whether the menu line definition of a form skin is used to draw menus when the value of the **myMenuSkin** property is null (**""**). If the value of the **useMenuLineSkinForMenus** property is:
  - **False** and the value of the **myMenuSkin** property is null (**""**), menus are not skinned
  - **True** and the value of the **myMenuSkin** property is null (**""**), menus are drawn using the menu line properties (that is, font, colors, and so on)

  Control the **useMenuLineSkinForMenus** property with the **Use Menu Line Options For Menus** check box on the **Forms** sheet of the Jade Skin Maintenance dialog. If a popup menu skin is set for the form, the **Use Menu Line Options For Menus** check box is set to **false** (that is, unchecked) and disabled.
- The **JadeSkinForm** class now provides the Boolean **drawMenuSelectionFlat** property, which controls how the menu line items are drawn when the skinned menu line item is selected. If the value is:
  - **False**, a selected menu line item is drawn using the effective menu item selection background and foreground (text) colors with a border that is the same color as the text of the menu item.

    In addition, when a menu is dropped down directly below a menu item (the menu does not have to be moved to fit on the current display), the selected menu item is drawn as though it is part of the dropped-down menu, using the same background color and text color defined for the popup menu.
  - **True**, the selected menu line item is always drawn in a flat manner using the effective menu item selection background and foreground (text) colors with no surrounding border.

    In addition, the menu line item is not drawn as though it is part of the popup menu.

  Control the **drawMenuSelectionFlat** property with the **Draw Flat Selection** check box on the **Forms** sheet of the Jade Skin Maintenance dialog.
- The **JadeSkinMenu** class now provides the Boolean **drawMenuSelectionFlat** property, which controls how menu items are drawn in the popup menu when a menu item is selected. If the value is:
  - **False**, the selected menu line item is drawn using the effective menu item selection background and foreground colors with a border that is the same color as the text of the menu item.
  - **True**, the selected menu line item is always drawn in a flat manner using the defined menu line selection colors (with no surrounding border).

  Control the **drawMenuSelectionFlat** property with the **Draw Flat Selection** check box on the **Menus** sheet of the Jade Skin Maintenance dialog.

Rollover Foreground Color (NFS 65850, JEDI 57)

The **JadeSkinWindowStateImage** class now provides the Integer **foreColor** property, which enables you to define the color of the text for the defined window state. This property is used for the text for a defined window state of a **Button**, **Folder** control sheet tab, and **JadeMask** control when a **JadeSkinButton** skin is assigned.

The default value of **#80000000** means that no foreground color is applied to the text color of the control by the window state so that the text color is defined by the value of the **foreColor** property of the **JadeSkinButton** class or **Control** class; otherwise the value is used to draw the text when a specific window state is active. For example, when the mouse is moved over a button and an **imgRollOver** value is defined for that state, the text is drawn using the **foreColor** property value when it is not **#80000000**.

This behavior is not affected by any **foreColor** property value assigned to the control. It would therefore normally not be used to draw the text in its normal state, because otherwise a skinned control will not show a **foreColor** value assigned by logic to the control.

The **Window State Image** sheet of the Jade Skin Maintenance form now contains the **Default foreColor** check box, which is checked by default. When you uncheck the check box, the common Color dialog is displayed, to enable you to select or define the foreground color you require for the **rollOver** state of the skin.

Specifying Some Skin Table Colors (NFS 65152)

The **JadeSkinTable** class now provides the following properties.

- **fixed3D**

  This **Integer** property controls whether the table fixed cells are drawn in three-dimensional style. The property can be set to one of the following values.

Integer Value	Description
0 (the default)	A table when it uses this skin does not draw the fixed cells as 3D elements. (The value of the **Table** class **fixed3D** property is ignored.)
1 (true)	A table when it uses this skin draws the fixed cells as 3D elements. (The value of the **Table** class **fixed3D** property is ignored.)
2 (use the **Table** control value)	Ignores this skin property and uses the value of the **Table** class **fixed3D** property.

- **fixedColumnsBackColor**

  This **Integer** property specifies the color with which the background of fixed columns is drawn.

  The default value of **#80000000** means that this property is ignored. The background color of fixed columns for any other value is drawn using the specified value of this property.

- **fixedRowColorHasPrecedence**

  This **Boolean** property specifies whether cells that are in both a fixed row and a fixed column are drawn using the value of the **fixedRowsBackColor** property. The default value is **true**.

  When **true**, cells that are in both a fixed row and a fixed column are drawn using the value of the **fixedRowsBackColor** property. When **false**, cells that are in both a fixed row and a fixed column are drawn using the value of the **fixedColumnsBackColor** property.

- **fixedRowsBackColor**

  This **Integer** property specifies the color with which the background of fixed rows is drawn.

  The default value of **#80000000** means that this property is ignored. The background color of fixed rows for any other value is drawn using the specified value of this property.

Note If a fixed cell has a specific background color set by the **Table** class **backColor** property value of a cell, row, or column, the background color of a table skin using one of the above properties is ignored.

Table Row Alternating Colors (NFS 65216)

The **alternatingRowBackColor** and **alternatingRowBackColorCount** properties have been added to the **JadeSkinTable** class (and to the **JadeSkinListBox** class). You can set these values in a **Table** skin using the functionality provided by the **JadeSkinMaintence** form. If the value of the **alternatingRowBackColorCount** property is zero (**0**), the value of the **alternatingRowBackColor** property is ignored and does not apply.

The skin settings are also ignored if logic specifically sets the value of the **alternatingRowBackColorCount** property of a table to a value greater than zero (**0**).

Note If a fixed cell has a specific **backColor** set via a cell, row, or column **backColor** property value, the skin back color value that applies above is ignored.

For details about the **alternatingRowBackColor** and **alternatingRowBackColorCount** properties, see "Alternating Colors for List Box and Table Rows (NFS 65216)", earlier in this document.

Text Box Hint Foreground and Background Colors (NFS 65859)

The **JadeSkinTextBox** control now provides the properties listed in the following table.

Property	Value	Default	Description
hintBackColor	Integer	#80000000 (use the defined **TextBox** class **hintBackColor** property value)	Background color of hint text
hintForeColor	Integer	#80000000 (use the defined **TextBox** class **hintForeColor** property value)	Foreground color of the hint text

These values apply only if the **TextBox** control class text box **hintText** property is not null (**""**), the hint text is displayed (that is, the text box is empty), and the **JadeSkinTextBox** class hint values are not **#80000000** (the default). For details about **TextBox** class hint text and colors, see "Text Box Hint Text and Color (NFS 65829, JEDI 117)", elsewhere in this document.

These property values are also ignored when the hint text is displayed and the text box text is disabled, in which case the text box is drawn in its disabled state (with the hint text still displayed).

Set the values for the **TextBox** control type on the **Controls** sheet of the Jade Skin Maintenance dialog. The default selection text (foreground) and background colors are **true**, by default; that is, the **Default hint foreColor** and **Default hint backColor** check boxes are checked.

When the selection text or background color is not set to the default value or you uncheck the **Default hint foreColor** or **Default hint backColor** 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 text box hints.

Splash Screen GIF Images (NFS 65786, NFS 65792, JEDI 116)

The animation displayed on the JADE startup splash screen can now be a GIF (Graphics Interchange Format) file instead of an AVI file.

The **AviFile** parameter in the [Jade] section of the JADE initialization file on the client node can now be set to a **.gif** file, which is positioned according to the value of the **AviPos** parameter in that section of the initialization file.

SQL Server 2016 Support (PAR 66028)

From JADE 7.1 and later, Microsoft SQL Server 2016 is supported for RPS nodes.

Microsoft SQL Server 2016 requires one of:

- SQL Server Native Client 11.0 or later
- ODBC Driver 13 for SQL Server

String Replacement Methods (NFS 65528, JEDI 2)

The **String** primitive type now provides the **replace__** and **replaceFrom__** methods, which have the following signatures.

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

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

Both methods return a new string that has had the specified replacement made. The receiver string has the substring specified in 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.

The **replaceFrom__** method raises exception 1413 (*Index used in string operation is out of bounds*) if the value specified in the **startIndex** parameter is less than **1** or it is greater than the length of the original string. Both methods raise exception 1002 (*Invalid parameter value*) if the value specified in the **target** parameter has a length of zero (**0**).

In JADE 2016.0.02 (Service Pack 1), these methods were implemented as **stringReplace** and **stringReplaceFrom** external methods in the RootSchema **jomsupp** system library. As these methods were provided in a service pack, they were not defined in the **String** primitive type in the RootSchema, to prevent name conflicts.

Note The new **String** primitive type methods will conflict with any existing **String** method named **replace__** or **replaceFrom__** defined in an earlier release.

Thin Client Disconnection (PAR 66130)

JADE now always uses its own internal combo box implementation.

This is transparent to any user unless screen-reading software is being used, in which case less information is read when interaction with the combo box occurs.

Type Methods (PAR 65759)

Both instance *and* type methods can now be defined as abstract, so you can have a type method on all of the classes inheriting from the superclass.

Unit Tests

This section describes the JADE Unit Test changes in this release.

Additional Progress Data Displayed (NFS 66224)

The Unit Test Runner form now displays **Failed Asserts**, **Passed Asserts**, and **Unhandled Exceptions** columns in the **Progress** table. (The test failure counter is incremented once for each failed test.)

Changing the Font Size (NFS 57157)

You can now increase and decrease the font size (that is, the typeface) displayed in the Unit Test Runner form. The **UIFontSize** parameter in the [JadeUnitTestRunnerUI] section of the JADE initialization file specifies the size in points of the font used to display information in the Unit Test Runner form. The value can be **10** (the default), **11**, or **12**.

» **To increase the size of the displayed font, perform one of the following actions**

- Select the **Font Bigger** command from the View menu.
- Press Ctrl+1.

The font size is then doubled (that is, increased by a factor of 100 percent).

You can repeat this action once more if you want the displayed font to be 200 percent larger than the default font (that is, this command is disabled when you have selected it twice in succession).

» **To decrease the increased size of the displayed font, perform one of the following actions**

- Select the **Font Smaller** command from the View menu. (This command is disabled when the text in the Unit Test Runner form is the default font size.)
- Press Ctrl+0.

The font size is then halved (that is, decreased by a factor of 100 percent).

You can repeat this action once more if you want the displayed font to be 200 percent smaller than the maximum increased font size and returned to the default font size (that is, this command is disabled when you have selected it twice in succession).

Logging the Call Stack in a Batch Unit Test (PAR 65836)

Call stack logging on exception and assert is now enabled by default when running unit tests in batch mode.

You can disable the call stack logging, by specifying **<options logCallStack="false"/>** in your unit test XML control file.

Profiling Unit Tests (NFS 65837)

The JADE Unit Test framework now enables you to profile tests in:

- The Unit Test Runner form, by selecting the File menu:
  - **Profile** command, to start and stop collecting unit test results.
  - **Report Profile** command, to output unit test results to a file. If there is an existing file, records are appended to the file.
- Batch mode, by adding the **profile=true** argument to the:
  - **jade** executable **JadeUnitTestGuiNoForms** or **JadeUnitTestBatch** application or the **jadclient** executable **JadeUnitTestBatch** application command line.
  - Control file specified after the **startAppParameters** argument in the **jade** or **jadclient** command line.

The [JadeUnitTestRunnerUI] section of the JADE initialization file can now also contain the **Profile** parameter, which defaults to **false**. This parameter is set only by using the **Profile** command in the Unit Test Runner form File menu.

Running All Unit Tests in Batch Mode (PAR 65202)

In releases prior to JADE 2016.0.02, 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**.

Unit Test Method Options and Output Messages (PAR 66232)

In earlier releases, there could be more than one instance of the **unitTestBeforeAll** and **unitTestAfterAll** method options defined on methods within a schema. This caused an inconsistency when methods with the **unitTestBeforeAll** and **unitTestAfterAll** options were run, depending on the context of the tests run.

Because of this, instances of **unitTestBeforeAll** and **unitTestAfterAll** method options can now be defined only on the base **JadeTestCase** class within a schema, which means that if any number of test methods are run from the schema, the **unitTestBeforeAll**- and **unitTestAfterAll**-marked methods are executed once only.

If multiple **unitTestBeforeAll** and **unitTestAfterAll** method options are defined on methods of the **JadeTestCase** class in a schema, the first instance of the method option is used and a warning is logged, stating that all other instances have been ignored.

If **unitTestBeforeAll** and **unitTestAfterAll** method options are defined on methods in subclasses of the **JadeTestCase** class, they are ignored and a warning is logged, stating they have been ignored.

The behavior of the messages resulting from the **JadeTestCase** class has now also changed. When a test from a new schema is run, a **JadeTestListenerIF** class message event occurs before the first test method and after the last test method in the schema is executed. The **start** and **end** messages now specify the schema or schemas being tested, which is more consistent and useful than in earlier releases when only the names of the first and last class to be tested were output.

Web Service Provider Validation (NFS 46895)

JADE now handles an integer overflow when processing a received Web services SOAP message so that an exception similar to error 11054 *(String or Binary property has exceeded its maximum defined length)* and 11055 *(Result of expression overflows Decimal precision)* now occurs. This new error is 11059 *(Result of expression overflows Integer precision)*, which outputs the property name and class in which the overflow occurred to the respective **Exception** class **errorItem** and **extendedErrorText** properties. If this error occurs, fix your Web service message to avoid invalid Integer values.

In addition, the **JadeErrorCodesWebService** global constant category now provides the **JADEWS_INTEGER_OVERFLOW** global constant, which has and Integer value of 11059.

Web Session Timeouts (NFS 65524)

Web sessions now use a dictionary keyed by session id, which results in a significant performance improvement when web sessions time out or have been manually deleted.

WebSocket Protocol

JADE now supports the WebSocket communication protocol.

The WebSocket protocol enables interaction between a web client (such as a browser) and a JADE server-side application, facilitating real-time data transfer from and to the server. This is made possible by providing a standardized way for the JADE server-side application to send content to the client without being first requested by the client, and allowing messages to be passed back and forth while keeping the connection open. In this way, a two-way ongoing conversation can take place between the client and server-side application.

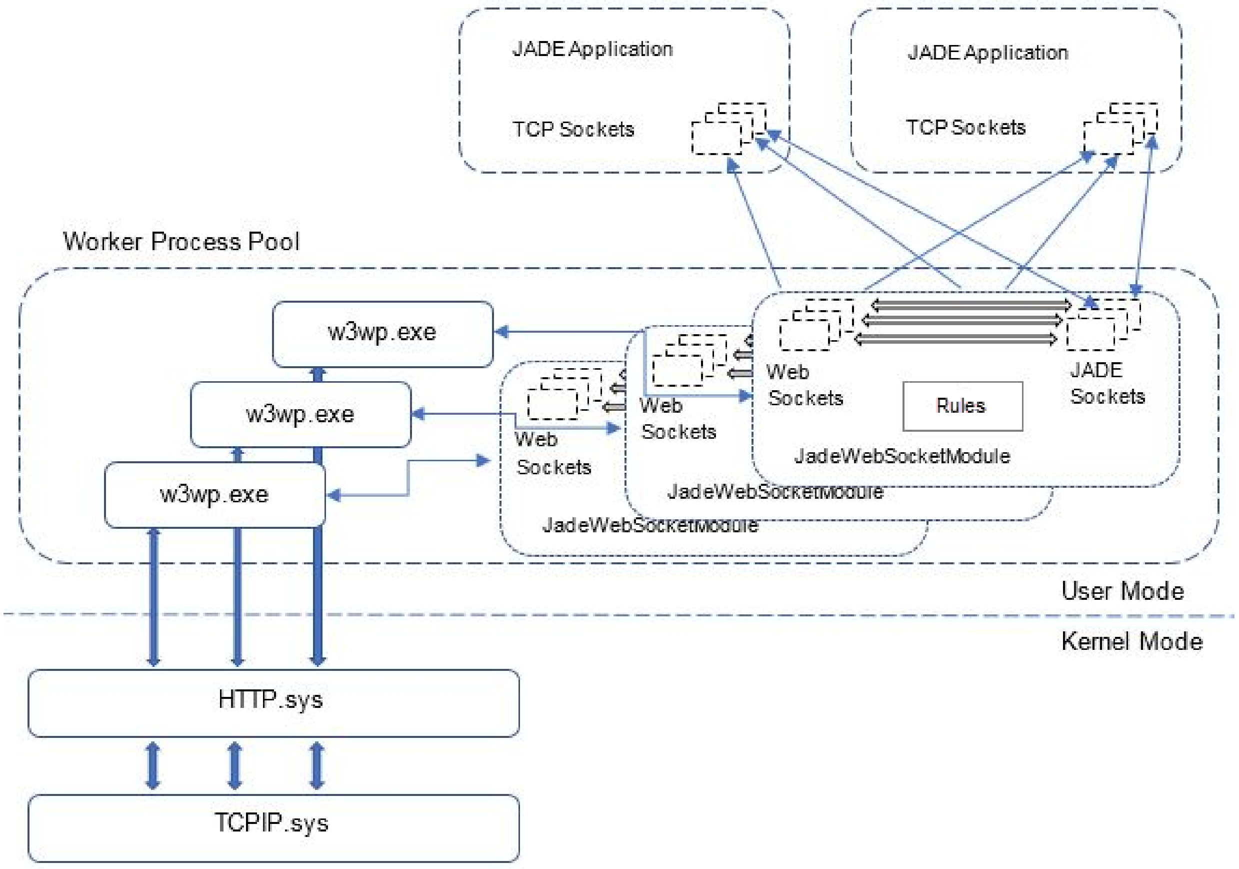
The communications are done over TCP port number 80 (or 443, in the case of TLS-encrypted connections), which is of benefit for those environments that block non-web Internet connections using a firewall.

IIS has to be correctly configured to handle incoming WebSocket connections and forward requests that will be handled by JADE to the **JadeWebSocketModule** (that is, the **jadeWebSockets_IIS.dll** native IIS module).

A **JadeWebSocketModule** instance loaded into an IIS worker process maps WebSocket connections to JADE application instances using URL pattern-matching rules that yield a host (network name or IP address) and TCP port number.

Note The WebSocket protocol is not supported by JADE applications connected to an Apache HTTP Server.

A simple message protocol is used to multiplex the WebSocket sessions and their messages over each JADE application TCP connection, as shown in the following diagram.



As IIS is typically configured to run multiple worker processes, multiple instances of the **JadeWebSocketModule** can be active at the same time, so JADE applications must support multiple TCP connections.

For details, see "Connecting to JADE Applications using the WebSocket Protocol", in Chapter 2 of the *JADE Installation and Configuration Guide*. See also the **JadeWebSocket**, **JadeWebSocketServer**, and **WebSocketException** classes in Volume 2 of the *JADE Encyclopaedia of Classes*.

Window Class Constants

The **Window** class provides the constants listed in the following table, which you can use to test the state of the Alt, Ctrl, and Shift keys of the **shift** parameter in the **keyDown** or **keyUp** event method.

Constant	Bit Value
KeyState_Alt	#4

Constant	Bit Value
KeyState_Ctrl	#2
KeyState_Shift	#1

XML in JADE White Paper

The JADE 2018 product information library now contains the *XML in JADE White Paper*, which enables you to use the XML framework in JADE for the rapid development of XML applications and XML components.

For more details, see the *XML in JADE White Paper*.