Upgrading to the JADE 2020 Release

VERSION 2020.0.02

Contents

Upgrading to the JADE 2020 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/jade-platform/developer-centre/release-policy.

JADE 2020 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.

Application Class Constants Deprecated

The following **Application** class constants have been deprecated in this release, and are now unpublished.

- ApplicationType_AGL_NoState_G
- ApplicationType_AGL_NoState_N
- ApplicationType_SilverLight

RPS Working Set Data Store Mode Deprecated

The RPS **Working Set** data store mode has been deprecated, as have the related:

- **JadeDatabaseAdmin** class **RpsStorageMode_WorkingSet** constant
- **AutoExtractOnPrimary<*n*>** parameter in the [JadeRps] section of the JADE initialization file
- Exception 3420 (*RPS reorg extract on primary failed*)

Highlights in this Release

The highlights in JADE release 2020, 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:

- Containerization, which provides a new way to run your JADE applications.

  Deliverables

  - Docker container-ready versions of backend components such as database and application servers as well as non-GUI standard client nodes.
  - Docker images for JADE services that can be used to configure and deploy a fully containerized JADE environment accessible from a public JADE Container Registry (JCR).

  Benefits

  Containers are efficient, secure, and portable. You can run existing JADE applications in a Docker-based ecosystem with no code changes.

  Value

  Containerizing your JADE application can help you meet some of the biggest challenges in IT: modernizing heritage applications, moving to the cloud, adopting DevOps, enabling coexistence with newer cloud-native applications, and staying innovative.

  For details, see "Containerization (Jad-I-433)", later in this document.

- Collection concurrency, which provides:

  - Conditional collection operations
  - New collection methods that allow updates to persistent collections to be deferred
  - New deferred update mode for automatic multi-valued inverses

  Benefit

  Provides a simple way to remove or avoid bottlenecks (and deadlocks) caused by contention for collection locks.

  Value

  Enables developers to improve the scalability of their JADE applications and remove code complexity around collection locking, thereby simplifying the code base.

  For details, see "Collection Concurrency (JAD-I-423)", later in this document.

- End-to-end Secure Sockets Layer (SSL) encryption between:

  - **jadehttp.dll** (IIS) and the application server (REST and SOAP web service providers)
  - SDS primary and secondary nodes

  For details, see "End-to-End SSL Encryption", in Chapter 2 of the *JADE Installation and Configuration Guide*.

- JADE REST client RootSchema classes, which allow for the easy consumption of REST services. A proxy class generator is also provided, which consumes an OpenAPI specification of a REST API and generates the appropriate data model and resource proxy classes.

  For details, see "REST Client (NFS 65594, JAD-I-97)", later in this document.

- The ability to restrict JADE REST APIs so that only clients with a valid bearer token can consume the API. In addition, an API developer can customize the rules on what constitutes a valid token to meet your requirements.

  For details, see "REST Service Security (JAD-I-430)", later in this document.

- The **JadeRegexLibrary** class, which is the abstract superclass of the regular expression (Regex) pattern-matching Application Programming Interface (API) subclasses in JADE. This API reduces hand-crafted string parsing and code manipulation, to assist in the reading and testing of your code.

  JADE does not implement a Regex engine itself, but wraps an existing implementation (the Perl Compatible Regular Expressions (PCRE) dialect) with defined behavior and documentation.

  For details, see "Regular Expression (Regex) Pattern Matching (JAD-I-438)", later in this document.

- The **foreach** instruction iteration using the new RootSchema interfaces **JadeIterableIF** and **JadeReverseIterableIF**, which provide contracts for an implementing class to support being iterated. These interfaces expose iterator implementations through new RootSchema interfaces **JadeIteratorIF** and **JadeReversibleIteratorIF**.

  For details, see "Iterating using Interfaces (JAD-I-571)", later in this document.

- The JADE Logical Certifier option that validates only inverse references and collections that have changed since the last time the system was certified.

  For details, see "Incremental Logical Certify (JAD-I-478)", later in this document.

- Many new features and changes in the JADE development environment; for example, a dark theme and the quick inspect functionality.

  For details, see "JADE Development Environment", later in this document.

Accessing Details about Faults Fixed in Releases

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

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

`https://www.jadeworld.com/jade-platform/developer-centre/support`

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 **2020** list item).

3. Click the **Search** button.

   Tip To display more than the default 100 entries returned by the search process, select the **User Preferences** command from the File menu to open the User Preferences dialog, select the **Search Defaults** command from the Searching submenu, and in the Maximum hits returned (latest search mode) group box, select the **All** option button or select the **No more than** option button and then increase the value (for example, to **300**) in the adjacent text box.

Upgrading to JADE 2020

This section covers the following topics.

For details about the JADE 2020 software requirements, which differ from those of earlier releases, see "Software Requirements", in Chapter 1 of the *JADE Installation and Configuration Guide*.

Caution Before you upgrade to JADE 2020, refer to "JADE 2020 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 2020 from JADE 2018

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 2020 from JADE 2018, by performing the following actions.

1. On the JADE *2018* 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 2018 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 2020 installer, available from https://www.jadeworld.com/jade-platform/developer-centre/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 2020 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 2020, 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 2020. 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 2020 Changes that May Affect Your Existing Systems

This section describes only the changes in the JADE 2020 release that may affect your existing systems. Some changes may result in compile errors during the load process, or cause your JADE release 2020 systems to behave differently.

.NET Import Issue Hotfix (PAR 67915)

Hotfix 2018.0.01.85 was released to fix some issues that were introduced with the new supported **DateTimeOffset** and **Byte[]** .NET types. (See also "Natively Supporting Additional .NET Imported Types in JADE", elsewhere in this document.) Any .NET assembly imported prior to JADE 18.0.01 continues to function as expected after applying this hotfix. However, as this was not introduced with the initial release of JADE 2018.0.01.85 may require you to run a fix-up script to correct the generated **JadeDotNetType** class methods and set compatibility flags.

Any JADE 2018.0.01 systems that imported or re-imported .NET assemblies before applying hotfix 2018.0.01.85 *and* have imported either of the following are required to run the provided fix-up script or to re-import the .NET assembly after applying the hotfix.

- Imported .NET classes with properties, fields, method or event parameters and /return values of the **DateTimeOffset** type
- Imported .NET classes with properties, fields, method or event parameters and return values of the **Byte[]** type

The RootSchema **Schema** class **_fixDotNetImportMethodsAndSetFlags** fix-up method needs to be executed once, after applying the hotfix if the system meets the above criteria. This script modifies and recompiles the necessary generated methods for affected assemblies, as well as sets some compatibility flags on the generated assembly meta data.

We have also provided another RootSchema **Schema** class **_fixDotNetImportCompatibilityFlags** method, which sets the compatibility flags. You should use this method if the method changes are deployed to the system.

Alternatively, any imported .NET assemblies that meet the above criteria can be re-imported after applying the hotfix. This causes the methods to be generated correctly and compatibility flags to be set.

The following is an example of running the fix-up method from the command line.

```
jadclient path=database-path ini=initialization-file-name
          app=RootSchemaApp
          schema=RootSchema
          executeSchema=RootSchema
          executeClass=Schema
          executeMethod=_fixDotNetImportMethodsAndSetFlags
```

Adding a String to a BinaryArray (PAR 67749)

Exception 1000 (*Invalid parameter type*) is now raised if the type of the value being added is not compatible with the type of the collection; for example, when attempting to add string values to a binary array.

This could occur if a local variable with type **Array** is used to send the **add** message to an array, because the compiler is unable to check the type of the parameter when the method is compiled.

Base URL for Context-sensitive Online Help

If you had specified a base URL (for example, **https://www.jadeworld.com/docs/jade-2020/Default.htm**) in the **JadeHelpBaseUrl** parameter in the [JadeHelp] section of the JADE initialization file, context-sensitive online help from JADE does not locate the topic to which you are navigating.

Context-sensitive online help is now provided when the following parameters in the [JadeHelp] section of the JADE initialization file are all set to **<default>**.

- htmlSchemes
- UseJadeWebHelp
- JadeHelpBaseUrl

If there is not a topic in the product information library for a JADE entity when you press F1, the HTML5 landing page (https://secure.jadeworld.com/JADETech/JADE2020/OnlineDocumentation/Default.htm) is displayed.

Collection Membership Length (JAD-I-84)

In earlier releases, an exception was raised when adding a value longer than the limit of 128 bytes for a **BinaryArray**; 63 characters for a **StringArray**; 2,048 characters for a **HugeStringArray**; and 30 UTF8 characters for a **StringUtf8Array**.

With the implementation of variable-size arrays in this release, an exception is now raised only if a value longer than the limit for the membership type is added to the array. The maximum supported length for the membership of these collections is now 16,000 for a **BinaryArray**; 15,999 for a **StringArray**; 15,999 for a **HugeStringArray**; and 8,000 UTF8 characters for a **StringUtf8Array**. An exception continues to be raised for user-defined subclasses of **Array** if a value longer than the defined length for the **Array** class is added to the array.

See also "Array Variable-Size Elements (JAD-I-84)", elsewhere in this document.

Dictionary Maintenance of a Key Property in a Collection (PAR 67895)

Dictionary maintenance is no longer performed if a property that is used as a key of a collection is updated but the new value is the same as the current value.

This change potentially improves performance for all JADE applications by avoiding unnecessary locking. Since the collection is not changed, update notifications are no longer sent. This means that collections passed to the **Table**, **ListBox**, or **ComboBox** control class **displayCollection** method with the **update** parameter set to **true** will not be refreshed if the key value for an object in the collection is updated but unchanged. To force the collection to be redisplayed, call the **Object** class **updateObjectEdition** method.

DDX-Format File Extraction (PAR 67676)

From JADE 2018.0.01, you could extract and load files in the XML Device Data Exchange (DDX) format, which extracted and loaded only those properties defined in the JADE Painter. The DDX format in JADE 2018.0.01 was based on the ability to identify system meta data references by name and the contents therefore did not include object OIDs. As a result, it did not include user references to other objects if they exist in the DDX extract.

From JADE 2020, the DDX extract process now extracts all defined non-virtual primitive properties and primitive array properties on controls. (Virtual properties on other included classes are not extracted.) This change means that if you load DDX files into JADE 2020.0.01 that were extracted in JADE 2018.0.01, there will be many differences in the contents of the DDX file.

Note DDX files extracted in JADE 2020.0.01 will not load into JADE 2018, because they are no longer backwards-compatible.

Detecting Orphaned DevControlProperties Instances (PAR 66131)

The **Meta Certify** or **certifyMeta** Logical Certify operation now checks for **DevControlProperties** instances that have no owner; that is, the parent property is null.

This situation could have been created due to a historic problem when a forms definition (**.ddb**) file that contained **Control** definitions with design-time properties is loaded into a JADE version that was greater than the system in which the definitions were extracted.

Caution After upgrading to JADE 2020, systems that use Painter design-time properties may report occurrences of the following error. We recommend running the generated repair, then another meta certify operation.

```
*** Error 98: owner is null DevControlProperties/1000486.1-
>DevControlProperties::parent (1000486.1 created 07 April 2002, 17:36:53)

FIX1: delete 1000486.1 CHECK: '1000486.1'.asOid.DevControlProperties.parent=null
```

Exceptions Accessing System-only Features

Exception 1192 (*Feature is restricted to system processes*) is now raised when you attempt to execute **Object** class **sendMsg**, **sendMsgWithIOParams**, **sendTypeMsg**, **sendTypeMsgWithIOParams**, **sendTypeMsgWithParams**, **invokeIOMethod**, **invokeMethod**, and **setPropertyValue** methods to access system-only features, as these are restricted to internal system processes only.

See also "Security Restrictions", elsewhere in this document.

Loading User Preferences from an Earlier Release (NFS 67599)

As documented under "Loading User Preferences from the Command Line (NFS 67599)", later in this document, you can now load your exported user preferences file using the command line.

The loading of an exported file of your user preferences no longer supports the older-style editor key bindings entries in your user preferences that were created in releases earlier than JADE version 2018.0.01. The loading of such files is therefore rejected when you import an earlier user preferences file and by a batch file load using the **jadclient** or non-GUI client **jade** executable command line.

You should therefore export your older user preferences to a file in JADE 2018.0.01 and higher before importing or loading them into JADE 2020.0.01 and higher.

Multiple activate/deactivate Events Generated for One User Action (PAR 67865)

Multiple **activate** and **deactivate** events can be generated for the one user action. This mainly occurs when an MDI child is floated; for example, clicking on an MDI child that is docked generates an **activate** and **deactivate** event for the floating form and then an **activate** event for the docked MDI form. In addition:

- If the user clicks on an MDI child in a position that is not a control that can have focus, the form is not activated. Clicking on a control that is allowed focus causes the form to be activated.
- When a floating form has focus and the users clicks on another standalone form, the floating form receives a **deactivate** event but the MDI frame does not.

The handling of **activate** and **deactivate** events has now changed to ensure that the following events occur once only.

- **deactivate** of the prior active form.
- **deactivate** of the MDI frame of the prior active form if it was an MDI child and the new form gaining focus is not the MDI frame or a child of the MDI frame.
- **activate** of the MDI frame if the form to be activated is an MDI child and the frame is not already active.
- **activate** of the form to be activated.

If another Windows or JADE application is activated while a JADE application is active, only **deactivate** events are generated for the deactivated JADE application.

If a JADE application is activated when another Windows or JADE application is active, only the **activate** events are generated for the activated JADE application.

Natively Supporting Additional .NET Imported Types in JADE

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

The .NET Import Wizard has been updated to natively support the following types.

- **DateTimeOffset** .NET type now maps to the **TimeStampOffset** JADE primitive type
- **Byte[]** .NET type now maps to **Binary** JADE primitive type

When importing a new assembly in JADE 2018 and higher, these type mappings are automatically applied.

When re-importing an existing .NET assembly, a compatibility warning dialog is now displayed. This dialog gives the option to re-import in compatibility mode, which keeps the existing mappings, or to re-import with the new mappings, which replaces any existing properties and method references with the new types.

Re-importing a .NET assembly with the new mappings can cause methods to be marked in error if they were referencing properties or methods with the old mapping to **JadeDotNetType**.

If you imported or re-imported any .NET assemblies with these new mappings *before* hotfix 18.0.01.85 is applied, see ".NET Import Issue Hotfix (PAR 67915)", elsewhere in this document.

Reverting a 2020 Thin Client to JADE 2018 (PAR 68048)

Until releases earlier than JADE 2020 have been patched and deployed, exception 14144 (*Thin Client Tcp connection read error*) is raised if you attempt to revert a JADE 2020 thin client to JADE 2018.

When all major releases earlier than JADE 2020 have been patched, the thin client will be allowed to be reverted to any prior release that contains the fix.

Security Restrictions

The following security restrictions are now in place.

- Content included in Report Writer scripts has been further restricted.
- Access to the Workspace window from the RootSchema **Schema Inspector** application is now disabled.
- In earlier releases, the Schema Inspector Utility did not observe development security.

  It now observes development security; that is, in production systems that have development security enabled, inspecting a class will be blocked if the development **jadeDevelopmentFunctionSelected** security hook function **inspectInstances** task denies access for that class.

  See also "Quick Inspect Toolbar (JAD-I-168)" and "Inspector Security", elsewhere in this document.
- JADE REST Service APIs can now be secured with JSON Web Tokens.

  See also "REST Service Security (JAD-I-430)", elsewhere in this document.
- A small JavaScript cross-site scripting (XSS) injection hole in the HTML generation functionality has been closed.

See also "Exceptions Accessing System-only Features", earlier in this document.

SslConfigurationTool Executable (PAR 68118)

Because **localhost** is no longer a practical host name for clients to connect to, the **SslConfigurationTool.exe** utility has been changed to reflect this.

The **TryForDefaultConfiguration** and **SelectCertificateFromUI** actions have been enhanced to take the **ServerHostname** and **ServerPortNumber** command line arguments. If **ServerHostname** is not specified, by default, the SSL configuration tool uses the fully qualified domain name of the computer.

String Type replace__ and replaceFrom__ Methods (PAR 67698)

In earlier releases, the **String** primitive type **replace__** and **replaceFrom__** methods raised exception 1002 (*Invalid parameter value*) if the value specified in the **target** parameter had a length of zero (empty) string value.

From JADE 2020.0.01, this behavior has changed so that this no longer causes an exception but instead returns the original receiver String.

Suspending Parent Alignment when Positioning Controls (NFS 68413)

In the JADE Painter, it was difficult to position docking controls to meet your requirements when the **alignContainer** and **alignChildren** properties are set.

To improve this situation, the JADE Painter Layout menu now provides the **Suspend Parent Alignments** command. When this command is unchecked (the default), the **alignContainer** and **alignChildren** properties of controls behave as normal.

When the command is checked, the **alignContainer** and **alignChildren** properties of controls are treated as though the property values are zero (**0**) so that no automatic alignment occurs, which enables you to position controls to meet your requirements. When you uncheck the command again, normal behavior is resumed and the required alignments are applied.

Notes If you attempt to save the form when the command is checked, a message box is displayed asking if you want to continue. If you indicate that the save process is to continue, the form is saved using the current control positions that could differ from those that apply when normal alignment operations are in effect. If you do not want to continue, the save process is cancelled.

Suspending alignment applies only to the **alignContainer** and **alignChildren** properties and it does not affect the behavior of the **parentAspect** property or the **StatusLine** control.

Changes in JADE Release 2020.0.02 (Service Pack 1)

This section contains details about product and documentation changes in JADE release 2020.0.02 (Service Pack 1).

For details about release 2020.0.01 (the first general release of JADE 2020), see "JADE 2020 Changes that May Affect Your Existing Systems" and "Changes and New Features in JADE Release 2020.0.01", elsewhere in this document.

Array Membership of Type Any

The **Any** primitive type can now be selected for the membership type of a user-defined **Array** class. Values added to the array can contain an object reference or any primitive value.

The maximum length of **Binary**, **String**, and **StringUtf8** values is as follows.

- **Binary** - 16,000 bytes
- **String** - 15,999 characters
- **StringUtf8** - 8,000 UTF8 characters

The RootSchema does not have a subclasses of **Array** with membership type of **Any**. If you require such an array, subclass the **Array** class in your user schema, selecting **Any** as the membership.

Checking for Orphan Subobjects in User Data Files

In the absence of a full logical certify of all instances of all user classes (for example, before upgrading to a higher JADE release), you can check for orphan subobjects in user data files by specifying the **orphanSubobjects** operation in the JADE Logical Certifier non-GUI application of the **jadclient** program; for example:

```
jadclient path=c:\jade\system ini=c:\jade\system\jade.ini schema=RootSchema
app=JadeLogicalCertifierNonGui server=SingleUser startAppParameters
operation=orphanSubobjects logDir=c:\jade\logs\certify
```

Collection Class Methods

This section describes the new **Collection** class methods in this release.

Conditional Methods

The **Collection** class now provides the following conditional methods.

- The **tryAddIfNotNull** method attempts to add the value specified in the **value** parameter to the collection if the value is not null and it is not already contained in the collection. It returns **true** if the value was successfully added; otherwise it returns **false**.
- The **tryRemoveIfNotNull** method attempts to remove the value specified by the **value** parameter from the collection if it is not null and it is contained in the collection. It returns **true** if the value was successfully removed; otherwise it returns **false**.

See also "tryCopy__ Method (JAD-I-642)", elsewhere in this document.

tryCopy__ Method (JAD-I-642)

In the next major release (tentatively 2022), we have introduced a new **Collection** class **tryCopy** method, which copies the values from the receiver collection to the specified target **toColl** collection that are not present in the target collection, and returns a reference to the target collection.

You could previously implement this action in JADE code by iterating the source collection and calling the **tryAdd** method on the target collection for each entry; however, the **tryCopy__** method implements this more efficiently in C++, and it also handles special cases such as copying to an external key dictionary more easily.

To make the functionality available in the 2020 service pack with low risk of conflicting with your own **Collection** methods of the same name, the method has been named **Collection** class **tryCopy__**. The next major release will provide an upgrade process to rename usages of **tryCopy__** to **tryCopy**.

Caution If you currently have methods named **tryCopy** in the **Collection** class hierarchy, we advise you to rename these ahead of the next major release. If your **tryCopy** functionality is essentially the same as the new **Collection** class **tryCopy__** method provided in this release, you can switch to using that instead.

This method is reimplemented in the **Collection** class **Dictionary**, **DynaDictionary**, and **Array** subclasses.

Compound Assignments

The JADE language now supports compound assignment operators, which are an extension to the existing **:=** assignment operation. They provide a shorthand syntax for assigning the result of an arithmetic or concatenation operation.

The compound assignment operators listed in the following table are now supported.

Compound Assignment Operator	Assignment
+=	Addition
-=	Subtraction
*=	Multiplication
/=	Division
^=	Power
%=	Modulo
&=	Concatenation

Compound assignment operators perform the operation specified by the additional operator on the left and right operands, then assign the result to the left operand. For details and an example, see "Compound Assignments", in Chapter 1 of the *JADE Developer's Reference*.

Converting a String Type to a Time Type (PAR 68119)

When converting a **String** primitive type to a **Time** primitive type, the valid time string is now as follows.

```
hh:mm[:ss.fff] [pre-or-post-noon-indicator]
```

This string is converted to **Time**. The rules that apply when casting a string to a time are as follows.

- The delimiter between time elements is a single non-alphanumeric characters.
- The hour element must be numeric in the range 0 through 24 (inclusive).
- The minute and second elements must be numeric in the range 0 through 59 (inclusive).
- The millisecond element must be numeric in the range 0 through 999 (inclusive).
- A term specifying pre- or post-noon (that is, case-insensitive **"am"**, **"a.m."**, **"pm"**, or **"p.m."**) is valid only if the hour element is less than or equal to 12.
- **"12:00 a.m.".Time** converts to **00:00:00**.
- **"12:00 p.m.".Time** converts to **12:00:00**.

Custom MenuItem Events (NFS 68101)

You can now add user-defined event methods for menu items, to handle populating or refreshing the state of each in a recursive manner in a similar way that you can controls. (See "Adding Methods to Your Subclasses Control", in Chapter 5 of the *JADE Developer's Reference*".) In earlier releases, only the **click** and **select** events were available.

To add a custom menu item event method, add an external method to the **MenuItem** class as follows:

```
event-name(parameter-list) is CallMenuEvent in jadpmap updating;
```

After you have defined the custom menu item, clicking on a menu item property of a form will include that menu item name in the menu item event list that is displayed.

You can then define the event logic. To take effect, this event must be called manually in logic; for example:

```
mnuAddCustomer.event-name();
```

If the menu item is deleted, the associated event methods (including any custom events) are also deleted.

Database Initial and Extent File Sizes (PAR 68240)

From JADE 7.0 and higher, the database file initial and extent size minimum and default values increased. The documentation has now been updated to state that the minimum value is **64K** and the default value is **128K** for both the extent size and initial size file attributes.

Deleting Subobject Dynamic Properties (PAR 61609)

The **JadeDynamicPropertyCluster** class now provides the **deleteSubobjectDynamicProperty** method, which deletes a subobject dynamic property definition and any subobject instances that may exist for the class on which the cluster is declared and any subclasses. This allows for deletion of an exclusive dynamic property when there are instances of the parent class.

Subobject instances include:

- Exclusive collection instances
- **Binary** and **StringUtf8** values for dynamic properties with a maximum length or a length greater than **540**
- **String** values for dynamic properties with a maximum length or a length greater than **539**

Note As this method can result in a large number of objects being deleted, consideration should be given to calling this method in its own transaction.

Subobject dynamic property definitions and instances can be deleted only if the class in which it is defined is not being used by any other process. If production mode is set, a subobject dynamic property can be deleted only in single user mode.

Extended create Instruction Expansion

In this release, the JADE language syntax has been expanded to make the extended **create** instruction compatible with more usages.

In addition to allowing the extended **create** instruction to be used as the right-hand side expression of an assignment statement, it can now be used in the following situations.

- As a parameter of a method call, provided that the formal parameter is **constant** or **input**.
- As the expression of a **return** instruction.
- As a parameter to a superclass parameterized constructor within a **create** method.

Exception 6801 (*Cannot assign to create expression*) is raised if you attempt to assign an extended **create** expression as the value of an **io** or **output** parameter.

See "**create** Instruction" in Chapter 1 of the *JADE Developer's Reference* for more details and examples.

Generating OpenAPIs from JADE REST APIs (Jad-I-635)

From JADE 2020.0.01, you can parse OpenAPI specifications, and now you can automatically generate them from JADE REST APIs. This enables you to automatically generate descriptions for all of the methods that are part of the API and any classes that are required by the API. In addition, you can customize descriptive elements such as the **info** section.

The new REST OpenAPI classes are summarized in the following table. (For details, see Volume 1 of the *JADE Encyclopaedia of Classes*.)

Class	Description
JadeOpenAPI	Abstract grouping class for classes relating to the JADE OpenAPI Generator
JadeOpenAPIGenerator	**JadeOpenAPI** subclass for generating OpenAPI specifications programmatically, as an alternative to the OpenAPI Generation wizard

JADE also provides an OpenAPI Generation Wizard that enables you to generate an OpenAPI specification from a JADE REST class and any additional documentation set in the wizard. To open the wizard for a REST class, click the **Generate OpenAPI** button on the **Web Options** sheet of the Define Application dialog. For details, see "Generating OpenAPI Specifications from JADE REST APIs", in Chapter 11 of the *JADE Developer's Reference*.

Iterating through Virtual Collections (PAR 67606)

The ability to iterate through virtual collections using the **Iterator** class **startAtObject** method has now been reinstated.

JADE Container Image Naming Convention (JAD-I-631)

Docker images for the JADE services are provided. The component name for each image is one of the following.

- *database-server*
- *application-server*
- *application-database-server*
- *non-gui-client*
- *gui-client*

JADE Images

The following table lists the description and entrypoint process for each container image.

Component Name	Description	Entrypoint Process
database-server	Database server	jadrapb.exe
application-server	Application server node	jadappb.exe
application-database-server	Application server with local database	jadappb.exe
non-gui-client	SOAP or REST web service	jadclient.exe
gui-client	Web application or GUI web service	jade.exe

The following table lists the image names for each JADE component. Image names are used in docker **pull** commands, to pull JADE container images from the JADE container registry; that is, **registry.jadeworld.io**.

Component Name	Image Names
database-server	registry.jadeworld.io/jade/database-server:20.0.02-x64-U registry.jadeworld.io/jade/database-server:20.0.02-x64-A
application-server	registry.jadeworld.io/jade/application-server:20.0.02-x64-U registry.jadeworld.io/jade/application-server:20.0.02-x64-A
application-database-server	registry.jadeworld.io/jade/application-database-server:20.0.02-x64-U registry.jadeworld.io/jade/application-database-server:20.0.02-x64-A
non-gui-client	registry.jadeworld.io/jade/non-gui-client:20.0.02-x64-U registry.jadeworld.io/jade/non-gui-client:20.0.02-x64-A
gui-client	registry.jadeworld.io/jade/gui-client:20.0.02-x64-U registry.jadeworld.io/jade/gui-client:20.0.02-x64-A

JADE Development Environment

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

AutoComplete Parameter Display for Translatable Strings (NFS 68109)

The JADE AutoComplete feature now displays the expected parameters for a translatable string that has defined parameters in the same way that it does when you enter the parameters of a method call.

In addition:

- The parameter usages in the constant value are also highlighted in red as well as the parameter name definition itself.
- External function calls in logic now display the method signature and highlighted current parameter in the signature.

Delta Searches (NFS 68089)

When you enter text in the **Search for Delta Id Containing** text box on the Delta Browser, pressing Enter now performs the search so that you no longer have to tab to the **Search** button or use the mouse to click it.

If a matching delta is found, the delta is selected. If the delta is not found, a bell sound will result.

Display of the Class in which a Method is Defined (NFS 68122)

When the **Show Inherited** command is selected in the View menu, it can be difficult to determine the class in which a method is defined when a class has a large number of superclasses and methods. The Class Browser, Interface Browser, and Methods Browser now provides additional information about a displayed method and for methods in the Methods List of a browser.

For the methods displayed in the:

- Editor pane, when the mouse is hovered over the method name at the start of the source, the JADE AutoComplete feature now displays bubble help for the method, which includes the schema and class to which the method belongs.
- Methods List in the JADE Hierarchy Browser and Methods Browser, when the mouse is hovered over an entry, bubble help now includes the schema and class of the method, as well as the method signature.

Finding a Class by Number (NFS 68102, JAD-I-592)

In JADE 2020.0.01, the Find Type dialog allowed partial matching on numbers (NFS 67605, JAD-I-574).

When you check the new **Find Class Number** check box, it now no longer clears the **Find** text box when it contains numbers only.

Inspecting Code in the Debugger (PAR 68357)

A number of changes and enhancements have been made when the debugger is at a breakpoint and when inspecting code in the debugger.

- When the debugger is at a breakpoint, if nothing is selected and you hover the mouse over a variable, the value of the variable is displayed when possible (including path expressions). No methods are executed.

  If an expression is selected, hovering over the selection attempts to execute the expression. The expression must be compilable within the context of the method and must return a value.

  If the compile and execution succeed, the returned value is displayed.

  Note It is your responsibility for the effects that the expression may have on the database and execution of the method logic.

- The Inspector dialog accepts expressions that can be compiled and executed. There are no restrictions other than the expression able to be compiled and return a value.

  Expressions longer than 100 characters are accepted, and it is your responsibility for the effects that the expression may have on the database and execution of the method logic.

  Note You can now resize the debugger Inspect Variable dialog, which enables you to enter longer expressions.

- The inspector value dialogs for both the primitive value display and modification can now also be resized to better display longer and multiple line values.
- The Local Variables window now includes **self** unless the method being executed is a type method.
- The debugger now accepts expressions as watches. Watches, however, are limited to 100 characters (stored in an identifier array where the key is a maximum of 100 characters).

  The debugger accepts only the initial entry of a new watch expression if it successfully compiles, executes, and returns a value. To enter a new watch, therefore, the debugger must be at a break position where the expression is valid and the objects involved are not null.

  When the watch is active, if it is not valid in another context, the value is displayed as **[Out of scope]**. If the expression cannot be executed because a variable is null, ****not Initialised**** is displayed.

  Note It is your responsibility for the effects that the expression may have on the database and execution of the method logic.

- Watches in the debugger are again restored if the watch window was restored when the debugger starts up.

Painter Hierarchy for Form Dialog (NFS 68385, JAD-I-573)

The JADE Painter Hierarchy for Form dialog now provides the following option buttons that control the way in which controls are ordered under their parent.

- **Order by Name**, which orders the control children by name (the default value)
- **Order by Top/Left**, which orders the control children by the top position and then the left position when two controls have the same top position, providing a visual association with the form layout

In addition, the description for each displayed form and control now includes the top and left positions and the height and width sizes in pixels; that is, (T:*top* L:*left* H:*height* W:*width*), as shown in the following example.

```
lblWebSite (T:18 L:12 H:21 W:60) (Label) 'URL'
```

Package Class Name Handling in the Find Type Dialog (PAR 68459)

If there are duplicate imported class names in a schema because multiple packages have the same class name, each class name copy is displayed in the Find Type dialog list box. Selecting any of the duplicate imported class names now displays the correct imported class rather than the same imported class.

In addition, the **mouseHover** event has been implemented for the Find Type dialog list box so that hovering over any entry with the mouse now displays the text returned by the **display** method for the associated object in bubble help.

Suspending Parent Alignment when Positioning Controls (NFS 68413)

In the JADE Painter, it was difficult to position docking controls to meet your requirements when the **alignContainer** and **alignChildren** properties are set.

To improve this situation, the JADE Painter Layout menu now provides the **Suspend Parent Alignments** command. When this command is unchecked (the default), the **alignContainer** and **alignChildren** properties of controls behave as normal.

When the command is checked, the **alignContainer** and **alignChildren** properties of controls are treated as though the property values are zero (**0**) so that no automatic alignment occurs, which enables you to position controls to meet your requirements. When you uncheck the command again, normal behavior is resumed and the required alignments are applied.

Notes If you attempt to save the form when the command is checked, a message box is displayed asking if you want to continue. If you indicate that the save process is to continue, the form is saved using the current control positions that could differ from those that apply when normal alignment operations are in effect. If you do not want to continue, the save process is cancelled.

Suspending alignment applies only to the **alignContainer** and **alignChildren** properties and it does not affect the behavior of the **parentAspect** property or the **StatusLine** control.

Workspace Refactoring (PAR 68352)

The Refactor menu is disabled for a **Workspace** method because a **Workspace** has no **self** class object and therefore the menu is not relevant.

JadeDbFilePartition Class drop Method (PAR 62426)

The following caveat has been documented for the **JadeDbFilePartition** class **drop** method, which removes objects in the global partition index, removes the partition, and marks it as deleted.

Caution The **drop** method does not execute destructors or trigger inverse maintenance, which means that inversed collections that referenced objects in the partition will contain invalid object references after the **drop** method has completed.

MergeIterator Class startKey Methods (PAR 68511)

If the dictionaries attached to a **MergeIterator** object do not have the same keys, the values specified in the **keys** parameter of the **MergeIterator** class **startKeyGeq**, **startKeyGtr**, **startKeyLeq**, and **startKeyLss** methods must be the same as the subset of common keys of the attached dictionaries.

Reorganization

This section describes the JADE database reorganization changes in this release.

Initiating a Managed Reorganization Transition

Product information of prior releases did not document the **JadeReorgApp** application **initiateManagedTransition** action.

This action initiates the offline phase of a reorganization on the primary system and schedules the initiation of the transition on a secondary database by replaying the online phases of the reorganization that took place on the primary, stopping before the transition is initiated. The secondary system remains available and the inquiry applications continue to run.

Tracking is restarted at a time scheduled by the administrator on the secondary system. This action automatically shuts down the inquiry applications and server applications, and the offline phases of the schema instantiation are carried out.

For details, see "Initiating a Managed Transition", in Chapter 14 of the *JADE Developer's Reference*.

Reorganizing Changed Array Definitions (PAR 68306, PAR 683549)

Reorganization now supports some changes to the definition of an array.

Changing the definition of an array may require all instances of that dictionary, both exclusive and shared, to be reorganized.

Note You can change the definition for an array only if the membership type is one of **Binary**, **Decimal**, **String**, or **StringUtf8**.

The supported changes are:

- Decreasing the maximum length of an array definition with membership **Binary**, **String**, or **StringUtf8**. All values must be less than or equal to the new maximum length. Values that are longer than a decreased maximum length cause the reorganization process to fail and an exception to be raised.
- Decreasing the precision or scale factor of an array definition with membership **Decimal**. All values must be less than or equal to the new precision. Values that are longer than a decreased precision cause the reorganization process to fail and an exception to be raised.
- Enabling scaling of entries of an array definition with membership **Decimal**.

A reorganization is not required if the:

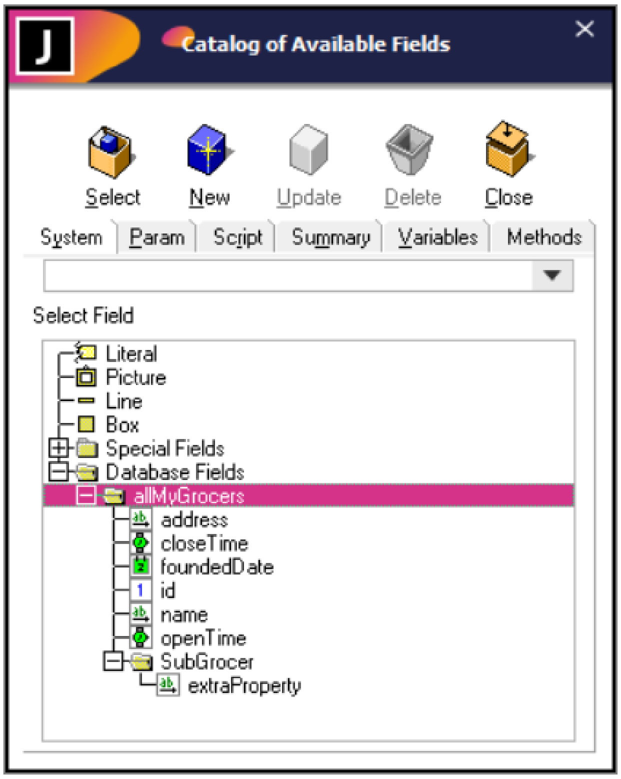
- Maximum length of an array with membership **Binary**, **String**, or **StringUtf8** is increased.
- Precision or scale factor of an array with membership **Decimal** is increased, or if scaling of entries is disabled.

Report Writer (JAD-I-552)

The following changes have been made to the JADE Report Writer in this release.

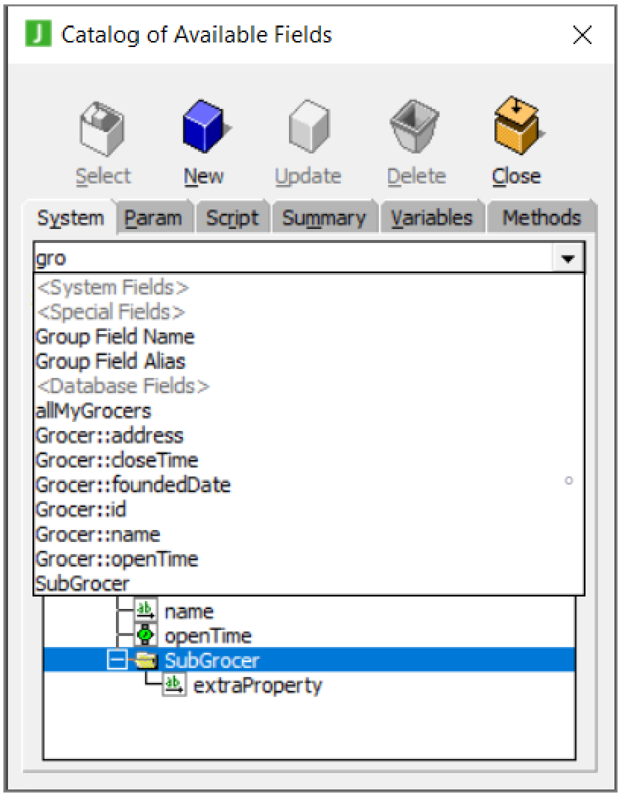
- Ability to copy and paste fields (for example, literals, database fields, and so on) between sections in the same report or different reports with the same root collections.

- The **Systems** sheet of the Catalog of Available Fields dialog now groups attributes and collections under their classes and allows the class to be collapsed, as shown in the following image.



In addition, the Catalog of Available Fields dialog now provides a combo box that enables you to quickly search for fields. The F4 function key also enables you to start your search.

All fields in the list box are searched; not just exposed classes, as shown in the following image.

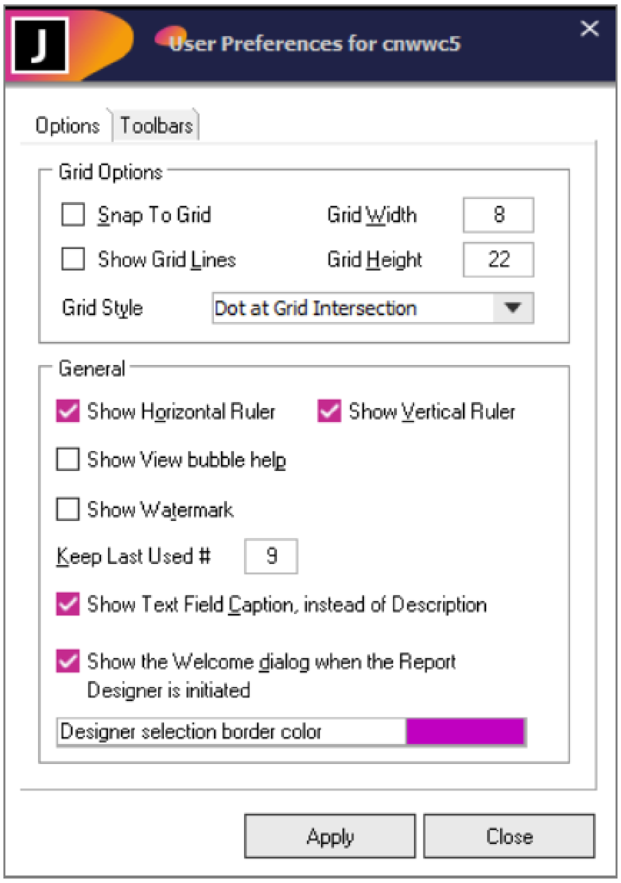


gro
<System Fields>
<Special Fields>
Group Field Name
Group Field Alias
<Database Fields>
allMyGrocers
Grocer::address
Grocer::closeTime
Grocer::foundedDate
Grocer::id
Grocer::name
Grocer::openTime
SubGrocer

name
openTime
SubGrocer
extraProperty

- You can now change the color of the dashed border around the currently selected fields on a report so that they stand out. The default border color is now purple (it was blue in earlier releases).

The **Designer selection border color** control on the **Options** sheet of the User Preferences dialog, shown in the following image, enables you to configure the border color of selected fields on a report to a color of your choice.


User Preferences for cnwwc5

Options | Toolbars

Grid Options

- ☐ Snap To Grid — Grid Width: 8
- ☐ Show Grid Lines — Grid Height: 22
- Grid Style: Dot at Grid Intersection

General

- ✓ Show Horizontal Ruler — ✓ Show Vertical Ruler
- ☐ Show View bubble help
- ☐ Show Watermark
- Keep Last Used #: 9
- ✓ Show Text Field Caption, instead of Description
- ✓ Show the Welcome dialog when the Report Designer is initiated
- Designer selection border color

REST Request PDF Data Format (PAR 68114)

The **JadeRestRequest** class now provides the **DataFormat_PDF** class constant, which has an integer value of **27** and corresponds to the **application/PDF** HTTP content-type. When using this format, set the **body** parameter as required by the REST API specification.

Running a Workspace in a Deployed System (PAR 68293)

If your JADE system has no developer licenses or has the JADE schema database files marked as offline, you can now run encrypted Workspaces supplied by JADE Support.

The **jadclient** executable program now provides the **executeEncryptedWorkspace** command, which has the following syntax.

```
jadclient ini=jade-initialization-file
          path=database-path
          executeEncryptedWorkspace=file-name
```

The encrypted Workspace is read from the specified file, which must have been provided by JADE Support. (Use of this command in **jadclient** is equivalent to opening an encrypted Workspace file and executing it in the JADE development environment.)

The **schema** and **app** commands are not required, and are ignored if they are specified.

The encrypted Workspace can be run in single user or multiuser mode, depending on the requirements of the Workspace.

Security (JAD-I-431)

SSL-enabled connections now include all client node to database connections, that is:

- Application server to database server
- Standard client to database server
- .NET client to database server
- ODBC standard client to database server

These connections are controlled with the **ClientToRap** parameter in the [Connections] section of the Secure JADE SSL configuration file.

Status Line Positioning (PAR 68131)

There was a conflict between the value of the **StatusLine** control **autoSize** property set to **true** and the use of the **parentAspect**, **relativeLeft**, and **relativeWidth** properties on the child controls. When the value of the **autoSize** property is **true**, the status line first positioned any children that are not fully visible, if possible. The parent aspect and **relativeLeft** and **relativeWidth** of the child control is then invoked but on the repositioned values. (Note that the relative properties are ignored when the **parentAspect** property is used.)

This behavior has now been changed so that **parentAspect**, **relativeLeft**, and **relativeWidth** properties used by child controls of a status line control work as expected regardless of the value of the **StatusLine** control **autoSize** property value.

When the value of the **StatusLine** control **autoSize** property is **true**:

- If a child control **left** property value position is less than zero, the control is moved to be zero (**0**) when the **parentAspect**, **relativeLeft**, and **relativeWidth** property values of the child do not affect the horizontal position (not stretch horizontal, anchor right, and centered horizontal, and the **relativeLeft** and **relativeWidth** property values are **false**).
- If a child control is not fully visible horizontally, the child is right-aligned in the status line control if it can be fully displayed or positioned at zero (**0**) if it cannot when the **parentAspect**, **relativeLeft**, and **relativeWidth** property values of the child do not affect the horizontal position (not stretch horizontal, anchor right, and centered horizontal, and the **relativeLeft** and **relativeWidth** property values are **false**).
- If a child control top position is less than zero (**0**), the control is moved to be zero when the **parentAspect** property value of the child does not affect the vertical position (not stretch vertical, anchor bottom, or centered vertical).
- If a child control is not fully visible vertically, the child is bottom-aligned in the **StatusLine** control when the **parentAspect** property value of the child does not affect the vertical position (not stretch vertical, anchor bottom and centered vertically).
- The values of the **relativeTop** and **relativeHeight** properties of child controls are always set to **false**, as their functionality is not compatible with auto-sizing the height of the **StatusLine** control (as has always been the case).
- All **parentAspect** flag values and **relativeLeft** and **relativeWidth** values are applied.

The height of the **StatusLine** control is then determined by analyzing the child control as follows, to determine the maximum height required. For a child control that does not have a fixed height and has the **parentAspect** property with the:

- **ParentAspect_StretchBottom** flag set, the height required is the **top** position, **height**, and **parentBottomOffset** property values of the child.

- **ParentAspect_AnchorBottom** flag set, the height required is the **height** and **parentBottomOffset** property values of the child; otherwise, the **height** of the child control.

These changes mean that the **parentAspect** property can be successfully used to position controls within a status line control.

Note The changes documented previously for PAR 68095 are no longer relevant.

Unicode Surrogate Pair Character Support (PAR 68066)

JADE now supports 16-bit Unicode code characters that are greater than **0xFFFF**, which includes emojis, mathematical symbols, and others.

The term *emoji* in JADE product information includes any type of surrogate pair characters. Such characters:

- Are stored as *surrogate pair* characters.
- Require the use of two 16-bit values to store the value. These Unicode values are stored as two encoded characters.

The actual value of the original Unicode character is split into two and stored in the lower part of each of the two characters, with **0XD800** added to the first 16-bit character and **0xDC00** added to the second. For example, the Unicode value **0x1F783** that represents an emoji is stored as **0xD83c** and **0XDF83**.

JADE now handles surrogate pair characters; in particular, the conversion from **String** to **StringUtf8** and **StringUtf8** to **String** primitive types. As a result of this, emoji characters can be included in text in a Unicode JADE system with some limitations.

The following code fragment is an example of Unicode surrogate pair character handling.

```
strUtf8 := #[f0 9f 8e 85].Binary.StringUtf8; // Father Christmas emoji
str := strUtf8.String;
write str;
write strUtf8;
```

A Unicode UTF-8 encoding table and emoji characters can be found at https://www.utf8-chartable.de/unicode-utf8-table.pl?start=127872.

The following is a list of notes about surrogate pair emoji usage in JADE.

- Emojis can be used only in a Unicode JADE system because they cannot be represented in ANSI.
- Emojis can be represented using **StringUtf8** in an ANSI JADE system, but they cannot be converted to a **String** value as there is no ANSI representation.
- Emojis can be included in any text displayed or printed, except for the rich text restriction later in this list.
- Emojis can be copied and pasted to and from **TextBox** controls or pasted into the editor pane.

  The emoji selection window (displayed using the Windows key + period (.), or dot, character key combination) can be used to paste a selection.
- Emoji characters can be converted to and from a **StringUtf8** primitive type, which means that can be included in web text data.
- An emoji character cannot be stored in a **Character** property because it requires two characters, so it will not fit.
- Emojis can be made up of multiple surrogate pair values that are overlayed on each other to get the final displayed representation. Using an arrow key can therefore leave the cursor positioned in the same place and may require multiple presses to step over each part of the emoji.
- The JADE length of a string that includes emojis is the number of 16-bit characters, *not* the number of displayed characters.

- Locate emoji characters in text by checking whether the first character is greater than **0xD800** and the second character is greater than **0xDC00**.
- The **JadeRichText** control, which is a Microsoft control, does not support emoji characters.

Unit Test Runner Form (NFSes 65290, 65287)

In this release, the Unit Test Runner form has been updated to include some new usability features.

The following changes have been made to the **Select Tests** pane of the form.

- It now displays the test classes in a hierarchical list. A top-level item is displayed for the current schema and subitems are added for all test classes in the schema.

  Individual tests are still represented as leaf items.
- The number of tests for each class is now displayed beside each test class. The total number of tests for a schema is also displayed beside the top-level schema item.
- If you select a class that has no local tests but it has subclasses with unit tests, clicking the **Run** button runs all tests from all subclasses.
- The text color of individual tests now reflects their status; that is, they are displayed in blue if they are yet to be run, green if the test succeeded, and red if the test failed.
- Double-clicking a method or class in the **Select Tests** pane now moves the focus to the JADE development environment and navigates to the selected entity in the relevant Hierarchy Browser or opens a new browser if one does not already exist for the current schema.

Other changes made to the Unit Test Runner form are as follows.

- The **Refresh** command in the View menu (or pressing F5) now runs or reruns all currently selected tests.
- New **Expand all** and **Collapse all** icon buttons have been added to the top right corner of the **Select Tests** pane.
  - The **Expand all** button expands the list box, making all tests visible.
  - The **Collapse all** button collapses the list box, showing only the schema item and its direct children. This represents all direct children of the **JadeTestCase** class.

Changes and New Features in JADE Release 2020.0.01

This section summarizes the product and documentation changes and new features in JADE release 2020.0.01. For details about the changes in release 2020 that may affect your existing systems, see "JADE 2020 Changes that May Affect Your Existing Systems", earlier in this document.

Animate Window Functionality for Forms (NFS 67576)

The animate window functionality is now provided for forms rather than just for controls, as it was in earlier releases. This allows for the display of an animated informational popup window without having to write a significant amount of logic, for example.

The **Window** class now provides the **animateWindow** method, which has the following signature.

```
animateWindow(millisecs: Integer; animateType: Integer) clientExecution;
```

This method is the superclass implementation of the **Control** class **animateWindow** method, which has been marked as unpublished so that existing uses are unaffected.

The documentation is the same as it was for the **Control** class **animateWindow** method except as follows.

- The **Control** class constants used for this functionality have been renamed and added to the **Window** class. The new names have the prefix **AnimateWindow_Flags_** instead of **AnimateWindow_**; for example, **AnimateWindow_Flags_Slide**.

  The **Control** class constants have been marked as unpublished so that uses in earlier releases are unaffected.

- The following new **Window** class constants apply only to a top-level form.

  - **AnimateWindow_Flags_Activate** (integer value of **#20000**) causes the form to be activated when it is shown. If it is not set, the form is shown without it gaining focus. In addition, when not set, the form retains its current **zOrder** position if it is redisplayed using the **animateWindow** method.

  - **AnimateWindow_Flags_Blend** (integer value of **#80000**) causes the form to fade in or out (This setting cannot be used with other animation effect options.)

- The **animateWindow** method call on a window does not result in any animation being shown if the form is maximized, modal, or an MDI child.

- The **Window** class implementation shows the border and caption of a form, and animates the contents inside it. This does not always produce the best appearance, and Windows does not always draw the animation correctly. If the form:

  - Has no border and caption, the animation should work correctly and be more appealing.

  - Is skinned, the effects apply to the whole form, including the border area, and the animation is drawn correctly.

- If the form has not been shown, calling the **animateWindow** method causes the **load** method to be executed.

Notes The **animateWindow** method is available only in GUI applications and for JADE forms (that is, it is not available for Web forms).

As the **Control** class **animateWindow** method and its associated class constants have been superseded by the **Window** class **animateWindow** method and its associated class constants from JADE release 2020.0.01, they are now unpublished in JADE 2020 so that use in earlier releases is not affected.

Any Primitive Type Methods

The **Any** primitive type now provides the methods summarized in the following table.

Method	Returns...
getType	The type of the value that is assigned to the receiver
isIntegral	**true** if the receiver can be type-converted to an **Integer** primitive type without any loss of data
isIntegral64	**true** if the receiver can be type-converted to an **Integer64** primitive type without any loss of data
isNumericType	**true** if the type of the value assigned to the receiver is a **Decimal**, **Integer**, **Integer64**, or **Real** primitive type
isTextType	**true** if the type of the value assigned to the receiver is a **Character**, **String**, or **StringUtf8** primitive type

Application Methods

This section describes the changes to **Application** class methods in this release.

Allowing Zero Forms

The **Application** class now provides the **allowZeroForms** method, which continues the process after the **Application** class **startAppMethod** or **startAppMethodWithString** method completes, even if a form was not created.

The **allowZeroForms** method does nothing if the application is running on the server or on an application server; that is, the application must be running on the client for the action to apply.

Code a **terminate** instruction to end the process when zero forms are allowed.

Starting an Application Method with a String

The **Application** class now provides the **startAppMethodWithString** method, which enables your logic to initiate another application on the same node as the initiating application, passing a string as a parameter for the specified method to be invoked on the new application.

Array Variable-Size Elements (JAD-I-84)

Arrays now provide efficient storage for **Binary**, **String**, and **StringUtf8** arrays. In earlier releases, array values were stored in fixed-size entries so that the length of the array membership needed to be set high enough for any eventuality. Arrays of maximum-sized values therefore resulted in a waste of resources.

The **Membership** sheet of the Define Class dialog now contains the **Maximum Length** check box. This check box is enabled if the membership is **Binary**, **String**, or **StringUtf8**. The length value specified in the **Length** text box cannot exceed the maximum supported length for the membership type, as follows.

- **Binary** - 16,000 bytes (maximum of 128 bytes in earlier releases)
- **String** - 15,999 characters (maximum of 63 characters in earlier releases)
- **StringUtf8** - 8,000 UTF8 characters (maximum of 30 UTF8 characters in earlier releases)

The **BinaryArray**, **StringArray**, **HugeStringArray**, and **StringUtf8Array** RootSchema classes are no longer defined as arrays of fixed-length values. The length of values in these arrays can now be up to the maximum supported length for the membership type.

See also "Collection Membership Length (JAD-I-84)" under "JADE 2020 Changes that May Affect Your Existing Systems", earlier in this document.

Button Picture Scaling (PAR 66907)

When an icon image is assigned to a **picture**, **pictureDown**, or **pictureDisabled** property of a button and the **Button** class **autoSize** property is not **AutoSize_Picture**, the size of the icon selected from the assigned icon image is now based on the client height of the button.

Collection Concurrency (JAD-I-423)

Locks on persistent collections create a bottleneck that affects most production multiuser systems in some way. Minimizing contention and avoiding deadlocks is a challenge for developers. To help with these challenges, JADE now provides:

- Conditional collection operations
- New collection methods that allow updates to persistent collections to be deferred
- New deferred update modes for automatic multi-valued inverses

The deferred execution model is a good choice when applied to collections that are updated but not read within a database transaction. Some of the benefits of deferred execution are:

- Deferred execution methods can be called at any point within the transaction. Since a deferred execution does not lock the collection, multiple processes can execute the deferred operations concurrently.
- Handles concurrent additions or removals of the same object executed by different processes in overlapping transactions.
- Provides a deadlock avoidance strategy.
- If application logic does not read the collection in the updating transaction, a shared-to-exclusive lock upgrade does not occur.

Notes The deferred add and remove operations are not visible to the calling process until after the enclosing transaction has committed.

The **ExternalCollection** and **JadeBytes** classes do not implement any of the **try*xxx*** methods. Attempts to call these raise exception 1068 (*Feature not available in this release*).

Deferred execution methods and non-deferred execution updating methods cannot be called on the same collection within the same transaction; otherwise exception 1471 (*Collection locking is incompatible with prior updates*) is raised (PAR 67879).

The **Process** class now provides the **useDeferredInverseMaintenance** and **overrideDeferredInverseMaintenance** methods, which support enabling or disabling deferred execution for all automatic or manual/automatic inverse collection properties for the current process.

The **Collection** class now provides the abstract **tryAdd** and **tryRemove** conditional methods, which are implemented by the **Array**, **DynaDictionary**, **ExtKeyDictionary**, **MemberKeyDictionary**, and **Set** classes.

The **Dictionary** class now provides the abstract **tryPutAtKey**, **tryRemoveKey**, and **tryRemoveKeyEntry** conditional methods, which are implemented by the **DynaDictionary** and **ExtKeyDictionary** classes.

Collection types that can contain objects provide **tryAddDeferred**, **tryPutAtKeyDeferred**, **tryRemoveDeferred**, **tryRemoveKeyDeferred**, and **tryRemoveKeyEntryDeferred** deferred conditional methods.

Deferred Execution Common Behavior

The following behavior is common to deferred execution.

- A deadlock avoidance strategy; that is, updates to multiple collections are grouped and processed in collection OID order
- Add and remove operations are processed in the order in which they were queued

Development Environment Changes

The Update Mode group box on the extended Define References dialog in the JADE development environment provides the **Deferred Execution** check box. You can check or uncheck this check box only when the **Automatic** or **Man/Auto** option button is selected (that is, it is disabled when the **Manual** update mode is selected or the corresponding **Type** is not a **Collection** or a **Collection** that does not support deferred update modes).

New .NET API Methods

JADE now provides .NET API methods for conditional and deferred collections, as follows.

- **JoobCollection** class methods
  - TryAdd
  - TryAddDeferred
  - TryRemove
  - TryRemoveDeferred
- **JoobDictionary** class methods
  - TryPutAtKey
  - TryPutAtKeyDeferred
  - TryRemove
  - TryRemove
  - TryRemoveDeferred
- **DynamicDictionary** class method
  - TryRemoveDeferred
- **ExtKeyDictionary** class method
  - TryRemoveDeferred
- **JoobSession** class methods
  - OverrideDeferredInverseMaintenance
  - UseDeferredInverseMaintenance
- **IJoobCollection** interface methods
  - TryAdd
  - TryAddDeferred

- TryRemove
- TryRemoveDeferred

These are documented in the JADE .NET API **JadeDotNetAPI.chm** file, which is in the installed JADE **documentation** directory (for example, **C:\Jade\JADE Docs\documentation**).

ComboBox and ListBox Class ItemNotFound Constant

The **ComboBox** and **ListBox** classes now provide the **ItemNotFound** constant, which has in Integer value of **-1**. For the:

- **ListBox** control, this indicates the value of the **listIndex** property when no item is currently selected (the default).
- **ComboBox** control, this indicates the value of the **listIndex** property when no item is currently selected or when the user has entered text that does not match any valid option in the combo box.

ComboBox Control Default Line Height (NFS 67273, JAD-I-467)

You can now specify the default line height of a combo box (for example, **cbt.defaultLineHeight := 25;**).

The **defaultLineHeight** property of the **ComboBox** class specifies the default height of lines in a combo box, independent of the font. This property represents pixels and defaults to zero (**0**), indicating that the height of a line in a combo box list is determined by the combo box font.

This property has no impact on the size of the combo box itself; only the height of the list items when the list is displayed.

File Open Dialog Prompt (PAR 65662, PAR 66871)

When using the common File Open dialog (the **CMDFileOpen** class), selecting a file, then navigating to another directory, you are no longer prompted to create the file selected, which prevents the display of unnecessary Create File dialogs. You are now advised that the file was not found, and prompting you to check the file name and try again.

The default value of the **CMDFileOpen** class **createPrompt** property has been changed to **false**. When a user now attempts to open a file that does not exist, the create prompt is not displayed and the user is no longer prompted to create the file each time a folder is navigated to from within the File Open dialog.

Note To achieve the behavior in earlier releases, you must now specify **cmdFileOpen.createPrompt := true;** before you call the **CMDFileOpen** class **open** method.

Containerization (JAD-I-433)

Containerization is defined as a form of operating system virtualization through which applications are run in isolated user spaces called containers, all using the same shared operating system. A container is essentially a fully packaged and portable computing environment.

- Everything an application needs to run – its binaries, libraries, configuration files, and dependencies – is encapsulated and isolated in its container.
- The container itself is abstracted away from the host operating system, with only limited access to underlying resources – much like a lightweight virtual machine (VM).
- As a result, the containerized application can be run on various types of infrastructure – on bare metal, within VMs, and in the cloud – without needing to refactor it for each environment.

Compared to a VM, there is less overhead during start-up and no need to set up a separate guest operating system for each application since they all share the same operating system kernel. Because of this efficiency, containerization is commonly used for packaging up the many individual microservices that make up modern applications.

The following subsections describe JADE containerization in this release.

Console Remote Access Program (jadrapb) (JAD-I-525)

In earlier releases, you can run a database server node as a GUI application using **jadrap.exe** or as a Windows service using **jadserv.exe**.

This release provides an additional **jadrapb** executable, which enables you to run a database server node as a console application. The main use case for this is to provide a docker container-ready entry-point process.

Run the console version of the JADE Remote Access Program from a command line, specifying the following.

```
jadrapb path=database-path ini=JADE-initialization-file-path
```

The following is an example command line.

```
jadrapb path=c:\jade\system ini=c:\jade\system.ini
```

The following is an example Dockerfile entry point specification.

```
WORKDIR /LogMonitor
SHELL ["c:/LogMonitor/LogMonitor.exe", "powershell.exe"]

# define the entrypoint process
ENTRYPOINT c:/jade/bin/jadrapb.exe ini=c:/jade/system.ini, path=c:/jade/system
persistentdb.journalrootdirectory=c:/jade/journals, jadelog.logfile=db_server,
jadelog.logdirectory=c:/jade/logs
```

Further Container-ready Services

The following service hosting console applications have been extended to operate correctly in Docker containers.

- **jadappb** – a console application that runs an application server node as a background process.
- **jadclient** - a console application that runs a standard client node as a background process. This is the preferred way to run a SOAP- or REST-based web service in a Docker container.

Container-ready processes shut down gracefully when the container is stopped by a user or a container orchestrator.

Docker Images

Docker images that can be used to configure and deploy a fully containerized JADE environment are served from the JADE Container Registry (JCR) [**registry.jadeworld.io**].

Each container image is configured to run a single JADE process. The code file variants cover Unicode, ANSI, and x64 variants.

Image Naming Convention

JADE container images are named according to the following convention.

```
registry-name/jade/component-name:TAG
```

The *component-name* is one of the following.

- *database-server*
- *application-server*
- *application-database-server*
- *non-gui-client*
- *gui-client*

The ***TAG*** is used to identify the version and build configuration, using the following format.

```
build-version-architecture-codeset
```

The ***TAG*** values are as follows.

- *build-version* is a JADE version string; for example, **20.0.01**
- *architecture* is **x64**
- *codeset* is **U** for Unicode or **A** for ANSI

The following is an example of the tag value.

```
20.0.01-x64-U
```

The following is an example of the full image name (or tag).

```
registry.jadeworld.io/jade/database-server:20.0.01-x64-U
```

JADE Images

Docker images for the JADE services are provided.

The following table lists the description and entrypoint process for each container image.

Component Name	Description	Entrypoint Process
database-server	Database server	jadrapb.exe
application-server	Application server node	jadappb.exe
application-database-server	Application server with local database	jadappb.exe
non-gui-client	SOAP or REST web service	jadclient.exe
gui-client	Web application or GUI web service	jade.exe

The following table lists the image names for each JADE component. Image names are used in docker **pull** commands, to pull JADE container images from the JADE container registry; that is, **registry.jadeworld.io**.

Component Name	Image Names
database-server	registry.jadeworld.io/jade/database-server:20.0.02-x64-U registry.jadeworld.io/jade/database-server:20.0.02-x64-A
application-server	registry.jadeworld.io/jade/application-server:20.0.02-x64-U registry.jadeworld.io/jade/application-server:20.0.02-x64-A
application-database-server	registry.jadeworld.io/jade/application-database-server:20.0.02-x64-U registry.jadeworld.io/jade/application-database-server:20.0.02-x64-A
non-gui-client	registry.jadeworld.io/jade/non-gui-client:20.0.02-x64-U registry.jadeworld.io/jade/non-gui-client:20.0.02-x64-A

Component Name	Image Names
gui-client	registry.jadeworld.io/jade/gui-client:20.0.02-x64-U registry.jadeworld.io/jade/gui-client:20.0.02-x64-A

Container Logging

Most container ecosystem logging solutions are built to pull from the **STDOUT** pipeline as is standard with Linux. Windows container application logs historically have not been accessible via these solutions.

JADE container images incorporate a Microsoft Open source tool called **LogMonitor**, sourced from GitHub at https://github.com/microsoft/windows-container-tools. The **LogMonitor** tool is configured to monitor **jommsg.log** output. Log output is written to **STDOUT** so that it can be accessed by the Docker engine or by log collection tools such as **Fluentd** or **Logstash**.

In addition, JADE container images are configured to write logs to a mappable internal directory that can be 'bind mounted' to an external directory on the host file system. This provides an additional simple way to persist and view **jommsg.log** output from outside a running container.

Windows Base Image

JADE container images use the Long-Term Servicing Channel (Windows Server 2019) base image https://hub.docker.com/_/microsoft-windows-servercore tagged **ltsc2019**.

The image is pulled from **mcr.microsoft.com/windows/servercore:ltsc2019**.

Image Update Policy

JADE base images will be updated to include cumulative hotfixes as they are released and on a regular basis to incorporate security and bug fix updates rolled out by Microsoft in updates to Windows server core base images.

Note As part of the containerized delivery way of working, we recommend that you put a process in place to ensure you update your image references to the latest base images on a regular basis.

Support Policy

JADE supports running JADE containers in on-premises configurations or cloud-based container platforms capable of correctly running Windows images based on **windows/servercore:ltsc2019**. If you experience issues or have questions about JADE container-related Docker functionality, JADE is your first point of contact.

For information about Microsoft's support policies for containers and related services on Azure, see:

`https://docs.microsoft.com/en-us/troubleshoot/azure/general/support-policy-containers`

For information about Microsoft's support policy for Windows containers and Docker in on-premises scenarios, see:

`https://support.microsoft.com/en-nz/help/4489234/support-policy-for-windows-containers-and-docker-on-premises`

JADE Container Examples

Several examples that demonstrate how to use JADE containers to stand up a variety of application environments are provided on a public GitHub repository at:

`https://github.com/jadesoftwarenz/JADE-container-examples`

The examples are all based on the Erewhon sample application. Each example has a **README** that explains how to deploy and run the example in a step-wise fashion as well as a single script that can be run to deploy a fully operational example environment on a laptop, PC, bare metal server, or in the cloud.

The example repository also contains documentation about how to get started using Docker Desktop for Windows and provides a script to automate the setup process.

Database

This section describes the JADE database changes in this release.

convertToBackup Command after Recovery (PAR 67837)

The **convertToBackup** operation is now permitted when the database image has been recovered to the end of a journal that contains in-progress transactions.

Note that it is not valid to run the **convertToBackup** operation on a database image:

- That is in a crashed state requiring restart recovery
- Where the class file mappings are inconsistent, which could happen if a step-wise recovery terminated during a schema instantiation

convertToBackup Command Extension (JAD-I-461)

Previously, the database **convertToBackup** function verified the integrity of the database and generated a **restoreinfo** file, creating the equivalent of a "restored backup".

The database **convertToBackup** function now creates a **backupinfo** file in addition to the restoreinfo file, creating the equivalent of a backup image that can be restored and recovered using standard database restore mechanisms. The result is both a backup and a "restored backup".

Database File Address Mismatch (PAR 67345)

In earlier releases, a database certify or backup action could result in logged warnings like that shown in the following example.

```
>>> Warning Coll header address mismatch - oid: [3924.49741942.2115.2.2:1], size=95
>>> address in index: 31892931552, actual file offset: 30049047889
```

Database certify warnings such as the above are now logged and counted as errors, as shown in the following example of a certify error.

```
06df4-1060 *** Coll header address mismatch - oid: [3924.49741942.2115.2.2:1],
06df4-1060     size=95 address in index: 31892931552, actual file offset:
06df4-1060     30049047889
```

Similar warnings detected by the database backup verification are now also logged as errors prefixed by **'***'**, but these are not counted as errors and do not result in a backup failure.

The database file certify sequential scan now includes the warning count in the summary; for example:

```
06df4-1060     34 verify errors, 94 warnings found scanning data
```

Database Diagnostic Enhancements (PAR 67280)

The JADE initialization file can now contain a [DBUtil] section, with parameters that provide new analysis capabilities. The parameters in this section are read when a file certify or freespace analysis operation is performed using:

- The JADE Database utility (**jdbutil** and **jdbutilb**)
- The JADE Database Administration utility (**jdbadmin**)
- JADE code with the corresponding **certifyFile** or **getFreeSpace** method on the **DbFile** class

In JADE 2018, Boolean values were disabled by default, to reflect existing behavior. However, the values of all but the **FetchAllObjects** parameter have changed in this release.

The parameters are listed in the following table.

Parameter	Default	Specifies...
FetchAllObjects	false	Whether each database object is fetched and certified. **Note** When set to **true**, this action may increase the time taken by the operation.
FreeSpaceDeepChecking	true	Whether enhanced checking of individual blocks in the freespace area is performed for **certify** and **freespace** operations.
FreeSpaceCrossChecking	true	Whether individual blocks in the separate freespace areas are checked for duplication in another area for **certify** and **freespace** operations.
FreeSpaceErrorLimit	0	The number of errors (for example, 20) that is reached during freespace analysis before the process stops and a message is logged to the specific operation log file (including why it stopped and the number of errors that were detected) when a **certify** or **freespace** operation is performed. The default value specifies that there is no limit to the number of freespace errors.

Date Primitive Type Methods

The **Date** primitive type now provides the methods listed in the following table, which enable you to determine information about days in a month.

Method	Returns...
daysInMonth	An Integer value equal to the days in the month of the date value of the receiver
lastOccurrenceOfDayInMonth	A date identical to the receiver except that the day is modified to the last occurrence that matches the specified day of the week
nthOccurrenceOfDayInMonth	A date identical to the receiver except the day is modified to match the specified *nth* occurrence (for example, 1st or 2nd) that matches the specified day of the week

Dynamic Objects

This section describes the JADE dynamic object changes in this release.

External Dynamic Object Methods (JAD-I-389)

Using JADE dynamic objects required you to make multiple get calls for the same property to retrieve different attributes of it, which could be computationally inefficient.

You can now implement the following **JadeDynamicObject** external methods.

Method	Description
getPropertyIndex	Returns the index of the property specified by the **name** parameter
getPropertyInfoByIndex	Outputs the name, type, and value of the property at the position specified by the **index** parameter
getPropertyInfoByName	Outputs the index, type, and value of the property specified by the **name** parameter

Merging Dynamic Objects

The **JadeDynamicObject** class now provides the **merge** method, which enables you to merge two or more dynamic objects.

Methods that Set Dynamic Object Properties

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

Method	Description
setPropertyValueAsPropertyType	Returns the index of the property specified by the **name** parameter
setPropertyValueAsPropertyTypeByIndex	Outputs the name, type, and value of the property at the position specified by the **index** parameter
tryGetPropertyValue	Returns the value of the specified property, if it exists

Processing and Obtaining Information about Dynamic Objects

The **JadeDynamicObject** class now implements the methods summarized in the following table, to provide you with more flexibility when obtaining information about and processing dynamic objects.

Method	Returns an implementation of the...
getPropertyNames	**JadeIterableIF** interface that enables you to iterate through the property names of the **JadeDynamicObject**
getPropertyTypes	**JadeIterableIF** interface that enables you to iterate through the property types of the **JadeDynamicObject**
getPropertyValues	**JadeIterableIF** interface that enables you to iterate through the property values of the **JadeDynamicObject**

Iterating through this interface using these methods returns the respective names, types, or values of the properties in index order on the dynamic object. (See also "Iterating using Interfaces (JAD-I-571)", elsewhere in this document.)

File Open Error Suppression (JAD-I-454)

The **File** class **tryOpen** method previously logged operating system errors in the **jommsg.log** file when the file open request failed for any of the following error causes.

- 5003 (*Requested file not found*)
- 5030 (*File is in use by another process*)
- 5040 (*Insufficient system resources*)

Operating system error logging is now suppressed.

firstVisibleLine Property Availability (JAD-I-449)

The availability of the **JadeTextEdit** control class **firstVisibleLine** has changed to read or write at any time. (In earlier releases, its availability was read-only at run time.)

Setting the value outside of the range **1** through the value of the **lineCount** property results in the value of the **firstVisibleLine** property being capped to that range.

Floating Form Visibility (PAR 67979)

When the **float** method was called on a docking control, the internal logic always made the floating form visible. (This was an incorrect assumption in that existing systems hide a docking control and then float it.) From JADE 16.0.01, this resulted in a change of behavior.

The JADE 7.1 behavior has been restored so that floating a docking control will not affect its visible status.

If the control is not visible, the floating form will not be visible. Changing the value of the **visible** property to **true** results in the display of the floating form.

foreach Instruction as Expression (JAD-I-173)

The **foreach** instruction now enables you to specify the optional **as** expression that, if present, contains an explicit type name or an expression that must evaluate to a type.

This enables the **foreach** target variable to be a subtype of the collection membership. Use this for variable type validation when you know that the collection contains objects that are more specific than the membership type of the collection; for example:

```
foreach dog in animalCollection as Dog do
```

Inspector

This section describes the JADE Schema Inspector changes in this release.

Inspecting a Deployed Database (PAR 66532)

The implementation of a deployed system changed in JADE 6.3, which resulted in the Schema Inspector application no longer calling the optional **allowedToInspect** security filter method to restrict access to user class instances. Since JADE 6.3, a *deployed system* means that the **_jadeapp** and **_jadedef** database files have been marked offline.

The Schema Inspector now blocks access to user class instances by default, if any of the following is true.

- Production mode is enabled
- The system does not have any developer licenses installed
- The **_jadeapp** or the **_jadedef** database file has been marked offline
- The **jadeDevelopmentFunctionSelected** security hook **inspectInstances** task fails for the selected class

You can implement the optional **allowedToInspect** method to override this behavior, or since the deployed JADE Database Inspector is subject to development security, you can implement the optional JADE development security hook **inspectInstances** task to control access. If none of these conditions apply, there are no restrictions to accessing user instances.

Notes If the **allowedToInspect** method returns **false** or the **inspectInstances** development security library hook fails for the selected class or classes, the inspection is denied; that is, a message box is displayed, informing the user that he or she does not have sufficient security rights to inspect instances of that class.

The quick inspect toolbar is never displayed in a deployed database.

Inspecting Class and Object Volatility (JAD-I-137)

The Schema Inspector now displays class and object volatility by default.

This enables you to check whether objects and collections are set to stable or frozen without having to write code to determine the volatility state of an object or collection.

Inspector Security

In earlier releases, development security covering the use of the Inspector form allowed unrestricted access to all instances.

- The **inspectInstances** security hook task is used by the quick inspect toolbar in both the JADE development environment and the Schema Inspector dialog to control access to the inspection of instances. The **entityName** input parameter enables you to control access to a specific class (*schema-name::class-name*).
- The **inspectMethod** security hook task toggles access to the **Object** class and **Collection** class **inspect** and **inspectModal** methods. The **entityName** input parameter enables you to control access to a specific class (*schema-name::class-name*).
- The **enableInspectToolbar** security hook task is used by the JADE development environment quick inspect toolbar and the Schema Inspector dialog to control access to the quick inspect toolbar itself. The **entityName** input parameter specifies the schema (*schema-name*) and application (*application-name*) that the quick inspect functionality can access. If an **enableInspectToolbar** security hook task fails in the:
  - JADE development environment, the quick inspect toolbar is disabled. If the quick inspect toolbar is already floating when this security hook fails, the floating quick inspect toolbar will be disabled but will still be able to be docked.
  - Schema Inspector, the quick inspect toolbar and the corresponding Option menu **Show Inspect Toolbar** command will not be visible.
- The **searchInstances** security hook task enables you to control the search behavior of the quick inspect toolbar both in the JADE development environment and the Schema Inspector dialog. The **entityName** input parameter specifies the schema (*schema-name*) to which to control access. When a user fails the security check in the:
  - Schema Inspector dialog, the quick inspect toolbar is still displayed, but accepts only valid OIDs.
  - JADE development environment, the quick inspect toolbar is still displayed but accepts only valid OIDs and the floating Quick Inspect form does not display a list box.

For details about mechanisms to restrict access to tasks in the JADE development environment and task authentication, see "JADE Development Environment Security", in Chapter 2 of the *JADE Object Manager Guide*.

Quick Inspect Toolbar (JAD-I-168)

The JADE development environment now provides the quick inspector toolbar, which:

- Removes the need to write code to inspect an object
- Adds the ability to search and view multiple instances of an entity

  The valid entries are an unqualified class (where only the current schema will be searched), an existing OID, a class number, or a qualified class name (that is, *schema-name::class-name*, *package-name::class-name*, or *schema-name::package-name::class-name*).

  Note If the JADE development security hook **searchInstances** task check fails, the only accepted entry is a valid OID.

- Can be docked in the development environment main toolbar, and contains:
  - A button (indicated by the **v** down arrow icon) that floats the control.
  - A combo box in which to enter the search entity or select a recently inspected entity. When no text is entered, the combo box displays the current schema.
  - The **Inspect** button that opens the Inspector form with the entity specified in the combo box.

  You can position the docked quick inspect toolbar anywhere in the main toolbar that does not overlap existing control icons (which are always left-aligned).

- Can be floated, by clicking the float button (indicated by the down arrow icon) or by dragging the toolbar off the main toolbar. The resizable Quick Inspect form that is then displayed contains:
  - A button (indicated by the **^** up arrow icon) that docks the control
  - A text box in which to enter the search entity
  - The **Inspect** button that opens the Inspector with the entity specified in the text box or selected in the list box
  - A list box that displays search results or previously inspected items for your current session

  The floating position and size of the Quick Inspect form is retained for the duration of your work session.

  When the number of entries in the list box exceeds 1,000, **Next**, **Previous**, **First**, and **Last** buttons are displayed to provide easy navigation over the list.

  When the text box is empty, the list box displays previously inspected items. (Historical items are preserved for the current session only.)

  If you specify a valid class number, the class name and instances of the class are displayed. If you specify a valid class name, the class number and instances of the class are displayed.

  The **allowedToInspect** security filter method called on a class enables you to define security to specify users who can inspect class instances in a deployed (runtime) database.

Tip Use the Ctrl+Alt+I shortcut keys to set focus to the search combo box when the quick inspect toolbar is docked or to the search text box on the Quick Inspect form when the quick inspect toolbar is floating.

If text is selected in more than one editor pane, the latest selected text is pasted into the corresponding search control. If multiple lines are selected, it is cut off at the first **CrLf** end-of-line sequence.

The JADE development environment main toolbar now provides a context (popup) menu that has the **Show Quick Inspect Toolbar** as well as the **Show Clipboard Toolbar** commands, which are checked by default. These commands are also provided in the development environment View menu. They toggle (hide or show) the display of the selected toolbar and update the value of the **Show Quick Inspect Toolbar** check box on the **Window** sheet of the Preferences dialog.

The **Window** sheet of the Preferences dialog contains the:

- **Show Quick Inspect Toolbar** check box (which is checked by default), to show or hide the display of the quick inspect toolbar. (If the quick inspect toolbar is docked in the toolbar of the main development environment window, hiding the main development environment window toolbar also hides the quick inspect toolbar.)
- **Quick Inspect Width** text box, which enables you to change the width of the quick inspect toolbar combo box from the default value of 300 pixels to the number of pixels you require, with a minimum value of 200 pixels and a maximum of 1,000 pixels.

The development environment Schema Inspector dialog:

- Now displays the **Inspect** search text box and button
- Replaces the existing hierarchy list box at the left of the dialog or form by a folder containing two sheets: the **History** sheet containing the existing hierarchy list box and the new **Search** sheet

Note The above list applies only to the development environment Schema Inspector dialog; *not* to the production Schema Collection Inspector form (that is, the Schema Inspector dialog started from the **JadeSchemaInspector** application in the **JadeMonitorSchema**).

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

In earlier releases, development security covering the use of the Inspector form allowed unrestricted access to all instances. For details about inspection security in this release, see "Inspector Security", elsewhere in this document.

JADE Development Environment

This section describes the JADE development environment changes in this release. (See also "Quick Inspect Toolbar (JAD-I-168)", earlier in this document.)

Adding a New Property (NFS 67555)

When you added a new property to a class and the Properties List displays the **Attributes**, **References**. **Controls**, or **Menu** sheet and the new property does not match the displayed type, the new property was not visible because it does not belong to the current list.

If the new property type does not match the type that is currently displayed, JADE now changes focus to the **All** sheet and the new property is visible and selected.

Application Default Inherited (NFS 67299)

If the default application is set to a superschema application on the Run Application dialog, the default application no longer reverts to the first application defined in the current schema.

The superschema application is now retained as the default and its name is displayed in the application name label at the right of the main (background) window of the JADE development environment browser.

When the Run Application dialog is displayed and the default application is a superschema application, the **Show Inherited** check box in the Inheritance group box of the Preferences dialog **Browser** sheet is automatically checked.

AutoComplete

This section describes the JADE AutoComplete feature changes in this release. (See also "Camel Case Filtering in List Boxes (NFS 67480, JAD-I-493)", elsewhere in this document.)

Block Label Prompt (NFS 66867)

The JADE AutoComplete feature now prompts you with the list of block labels (if any exist) when you type a character after a **break** or **continue** instruction.

In addition, a prompt is also displayed with the label associated with the block at that level (if one exists) when you type a character after an **endwhile** or **endforeach** statement.

Constructors with Parameters Signature Display (NFS 67275, JAD-I-426)

The JADE AutoComplete feature now displays the signature of the constructor when you specify a constructor with parameters in the editor pane, to indicate which parameter you are editing.

Displaying Identifier Usages from the Editor Pane (NFS 67553, JAD-I-517)

The JADE development environment and editor pane now enable you to select an identifier in the method source and request usages of the identifier by using the **Usages of *identifier*** command in the Edit context (popup) menu in the editor pane without having to find and select the item from a hierarchy browser and performing the same action.

Usages include references, property and method local references, property read and update references, method implementors, local implementors, and implementor references.

The cursor must be positioned in the identifier name with nothing selected or the whole identifier must be selected.

Note This functionality requires the JADE AutoComplete feature to be in use for the method (that is, the **Use AutoComplete** check box in the Auto Complete group box on the **Editor Options** sheet of the Preferences dialog is checked).

The identifier must be one of the following.

- Type name
- Class name
- Imported package class name
- Global constant
- Class constant
- Imported class constant
- Class property
- Imported class property
- Type or class method
- Imported class method

Display the Edit menu by selecting the Edit menu in the menu bar or by right-clicking in the editor pane of the method. If the cursor is *not* positioned on one of the above types of identifier, the 'usages' menu item is disabled and the caption displays *Usages - no suitable identifier selected*.

If the selected identifier is one of the following, the menu command that is displayed is **References of *identifier-type identifier-name***; for example, **References of Class Customer**.

- Type name
- Class name
- Imported package class name
- Global constant
- Class constant
- Imported class constant

Selecting the menu command displays the references to that identifier. If the identifier is a property or method, the displayed menu command has the caption **Usages of *identifier-type identifier-name***; for example, **Usages of Class Customer**.

Clicking the menu displays a submenu of the following options that depend on the type of identifier.

- **References**, for all property and method types
- **Local References**, for non-imported class properties and methods
- **Read References**, for property read references
- **Update References**, for property update references
- **Implementors**, for implementors of a method
- **Local Implementors**, for local implementors of a method if the schema type is a non-imported class
- **Implementor References**, for references to local implementors of a method if the schema type is a non-imported class

Clicking these commands performs the same actions as those when selecting the item in a browser list and then right-clicking and selecting the equivalent menu command.

If AutoComplete is not active, the **Usages of *identifier*** command is not displayed in the Edit menu.

Displaying Options for External Methods (PAR 67517)

The JADE editor now handles external methods in AutoComplete processing. It parses the definition of an external method, expecting it to be of the form:

```
method-name (method-parameters) [ : return-type] is entry-point-name in
library-name method-options;
```

Note AutoComplete does not offer any suggestions for the **is**, ***entry-point-name***, **in**, and ***library-name*** values.

AutoComplete cannot filter or check the validity of method options, but only check that the option is in the list of possible options.

Selecting an Exact Entry in the List (NFS 67276, JAD-I-468)

When the **Use History for Selection** check box on the **Editor Options** sheet of the Preferences dialog was selected, the AutoComplete feature automatically highlighted the word that was last selected. This could cause issues when you want to select an identifier that is a prefix of the selected word; for example, if the history was for **application** and you typed **app**, pressing the **.** key resulted in **application** being inserted when you required the **app** system variable.

The AutoComplete feature now selects an entry in the list when it is an exact match (case-sensitive) instead of the historical entry; that is, in the above example, **app** is selected when you enter **app**. The list item **application** is still selected when you enter **a**, **ap**, or **appl** in the editor pane.

Bubble Help

This section describes the bubble help changes in this release. (See also "Bubble Help in the Debugger (NFS 67666)", elsewhere in this document.)

Bubble Help in the Editor Pane (NFS 64066)

The information for the currently selected symbol that is displayed in the editor pane when you press F11 now includes the creation timestamp and patch version of the entity.

Displaying Shortcut Keys in Icon Bubble Help (NFS 67715, JAD-I-325)

When you hover the mouse over the icon on a toolbar button, the bubble help that displays the icon function now also displays the equivalent menu shortcut in the displayed bubble help text as *bubble-help-description* [*shortcut-keys*]; for example, **Browse Classes [Ctrl+B]**.

The shortcut key that is displayed for an action is set on the **Short Cut Keys** sheet of the Preferences dialog.

The development environment **Reorg**, **Context Help**, and **General Help** toolbar icons do not have an equivalent menu shortcut, so there is no shortcut displayed in the bubble help. To display the defined shortcuts in the bubble help for the other toolbar button icons, the following changes have been made.

- The list of shortcut combinations has been expanded in the **Shortcut Key** combo box on the **Menu Item** sheet of the Menu Design form in the JADE Painter. The combo box list now includes:
    - Ctrl+Alt+*alpha-character*
    - Ctrl+Alt+*numeric-digit*
    - Ctrl+Alt+*function-key*
- The development environment File menu now includes the following commands.
    - **Schema Extract** Ctrl+Alt+E (equivalent to clicking the **Extract** toolbar button)
    - **Schema Load** Ctrl+Alt+L (equivalent to clicking the **Load** toolbar button)

  Note The existing **Extract** and **Load** commands in the Schema menu of the Schema Browser are retained for user consistency.

- The File menu **New** command has a shortcut of Ctrl+Alt+W
- The File menu **Open** command has a shortcut of Ctrl+Alt+O
- The File menu **Print Selected** command has a shortcut of Ctrl+Alt+P
- The File menu **Monitor** command has a shortcut of Ctrl+Alt+M
- The Jade menu **Browse Schema** command has a shortcut of Ctrl+Alt+S
- The **History Back** and **History Next** toolbar button shortcuts have been changed to Ctrl+Alt+B and Ctrl+Alt+N, respectively. (During testing, it was found that the existing shortcuts of Ctrl+Alt+← and Ctrl+Alt+→ did not work, as Microsoft intercepts these sequences for its own desktop handling.)

Camel Case Filtering in List Boxes (NFS 67480, JAD-I-493)

In earlier releases, the editor pane and the Find Type dialog supported camel case filtering in list boxes only when the first sequence of uppercase characters of the list item match the filter text in that order.

This filtering process has now changed so that the search finds the uppercase sequence anywhere in the text of a list box item; for example, **CA** matches the entry of a **CMClass_Customer_Address** class. This avoids having to type any *XXXX* prefix you may have on all your class names, which reduces the number of characters that need to be typed to find a camel case match.

Captions on Floating Forms (NFS 67001)

When any of the JADE hierarchy browser major panels (the Class List, Properties folder, Methods folder, and editor pane) is floated, the floating form now has a caption, as follows.

- Class List panel displays *schema-name* classes; for example, **DocEgSchema classes**
- Properties folder panel displays *schema-name*::*class-name* properties; for example, **DocEgSchema::Reviewed properties**

- Methods folder panel displays *schema-name*::*class-name* methods; for example, **DocEgSchema::Reviewed methods**
- Editor pane displays *schema-name*::*class-name* editor; for example, **DocEgSchema::Reviewed editor**

Clipboard Toolbar Context Menu (JAD-I-168)

The JADE development environment toolbar now provides a context (popup) menu, which has the **Show Clipboard Toolbar** as well as the **Show Quick Inspect Toolbar** commands. These commands are also provided in the development environment View menu. They toggle (hide or show) the display of the selected toolbar and update the value of the **Show Quick Inspect Toolbar** check box on the **Window** sheet of the Preferences dialog.

Closing Tabs in the Development Environment (NFS 62115)

When the MDI option on the **Browser** sheet of the Preferences dialog is set to **Use Mdi With Tabs** or to **Use Tabs Only**, you can now close a tab by clicking the middle mouse button (that is, pressing the mouse wheel) on the tab. (The middle mouse button does not close pinned tabs.)

Notes This affects only the main development environment tabs.

You can still close a tab by clicking the close icon ☒ at the top right corner of the tab, including pinned tabs.

Comparing Method Sources (NFS 67704, JAD-I-160)

When comparing method source within a patch summary, the Compare Sources dialog now hides the merge editor pane by default, which provides more space for the displayed method sources.

A **Show Merge Pane** button is now provided, which when clicked, toggles the visible status of the merge editor pane. The button has the **Show Merge Pane** caption when the pane is hidden and the **Hide Merge Pane** caption when the pane is visible.

Notes The size of the merge editor pane is no longer saved and restored.

When the merge editor pane is visible, from JADE 2016, a resize bar enables you to drag the merge editor pane to reduce or increase its displayed size in the dialog.

Condition Method Text Template (NFS 67277, JAD-I-335)

When adding a condition method, the text template in the JADE editor no longer includes the **vars** and **epilog** sections if they had been included in the template.

Dark and Light Themes (JAD-I-370)

You can now set a pre-defined light or dark color mode in the JADE development environment so that you can select a dark theme if you prefer the dark color set that reduces excessive brightness. The JADE development environment also provides a default JADE theme and new dark and light skins.

A JADE theme is consists of:

- A name
- The Window colors
- The Editor colors
- The Relationship colors

The **Window** sheet of the Preferences dialog in the JADE development environment enables you to:

- Change between a light theme or a dark theme. (The **Editor** sheet and the sample list box in the **Window** sheet reflect the selected theme.)
- Select from dark and light versions of skins; for example. **JADE Lava Lamp Dark** and **JADE Lava Lamp**.
- Add, edit, reset, or remove a user-defined theme.

When you click the **OK** button on the Preferences dialog, the entire development environment is updated to reflect the selected theme.

The **Miscellaneous** sheet of the development environment Preferences dialog enables you to import or export a user-defined theme.

Debugger

This section describes the JADE Debugger changes in this release.

Bubble Help in the Debugger (NFS 67666)

The debugger has been enhanced to display the value of a path such as **sc.startDate**. This is now available in the following cases.

- If you hover the mouse over an identifier that is part of a path (for example, **startDate** in **sc.startDate**), the bubble help display is **sc.startDate = date**. The value is also displayed if the full path is selected.
- You can inspect a path in the debugger. If you place the cursor over a variable and then perform an inspect action, the Variable Inspector dialog is displayed with the full path name displayed.

  Notes If a valid path is selected in the editor text, the inspector value display is presented without the Variable Inspector dialog appearing.

  A path that contains a method call cannot be inspected.

- A path can now be used as a debugger watch expression.

Inspecting Variables in the Debugger (NFS 66542)

The Inspect Variable dialog in the JADE Debugger:

- Can now be resized. (There is also now a minimum height and width to which the dialog can be resized.)
- Can display more than the 40-character maximum value of earlier releases. The **Value** text box is resized as the dialog is resized.
- Enables you to directly modify the content of a variable, if applicable. The **Value** text box and **Modify** button are disabled if a variable cannot be modified; that is, it is an **Any** primitive type.

  The Modify Variable dialog has therefore been removed, as have the **Modify** context menu item in the Local Variables window and the **Modify Watch** context menu item.

Sorting Local Variables (NFS 67719, JAD-I-267)

In a method with a lot of local variables, it can be difficult to locate a specific variable to see its value.

The tables of displayed variables and watches shown by the JADE Debugger can now be sorted by clicking on the fixed column cell of the table.

By default, the variables in the Local Variables window are displayed in the order in which they appear in the method or as they are added to the Watches window.

Clicking in the fixed part or a column causes the entries to be displayed alphabetically sorted (case-insensitive) by that column for the rest of the debugger session. For example, clicking on an identifier in the **Name** column sorts the list by identifier name. Clicking the sorted fixed column cell again inverts the sorted list.

If the table is sorted, a numeric sorting indicator is displayed in the fixed cell of the column. If the column selected for sorting is *not* the identifier column, the identifier column is the second sort key.

Displaying Implementor References of a Method (PAR 66666)

The References to *schema::class:method* browser now includes the **Implementor References** command, which is also found in the JADE development environment Methods menu.

You can toggle the display of the methods list in the **+** (expand, or show) icon or the **-** (collapse, or hide) icon of the parent entry. Clicking on the entry displays only the name of the schema, the class in which it is defined, and the method name.

Displaying Local Methods Referencing a Property (NFS 67439)

The JADE development environment Properties menu now provides the **Local References** command. Select this command to add all local references of the selected property to the methods list in the pane on the Local References to *schema::class::property* browser.

This window lists all methods in the selected class and its subclasses that reference the selected property, and enables you to view specific references to that property.

Displaying Submenus (NFS 66558)

Left-clicking on a menu item that has a submenu immediately opens the submenu instead of waiting for the menu hover timer to expire before the submenu is displayed.

Filtering Combo and List Box Text Entries (NFS 67196, JAD-I-353)

Some JADE development environment and Painter forms now allow filtering of combo box and list box entries in a similar way to the existing F4 (Find Type) functionality, to enable you to locate a required list item more quickly by selecting from a list of suggestions for what is being entered in the current context.

When you enter text in a combo box or the text box associated with a list box, the displayed list items are only those entries that contain the specified text (case-insensitive) or if the specified text is all uppercase, the entries with the same Pascal case (for example, **CA** matches **CustomerAddress**). Because the search process handles uppercase searches if all text entered after the first character is uppercase, the list 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 of the navigation bar includes method names **searchNameTextOnly** and **someNumberedTest**.

This functionality applies to the following forms and controls.

- Painter Load Form dialog **Form** list box.

  When editing a form, the Load Form dialog provides the **Show as Hierarchy** check box, which defaults to **false** (that is, unchecked). When unchecked, the list is displayed alphabetically sorted and it allows the filtering process to be performed. When it is checked, the list is displayed in hierarchical form, as was the case in earlier releases, and the filtering process is not performed.

- Painter Find Control dialog **Control to Find** combo box.

- Define Reference dialog **Type** combo box (only for the current class on the left of the extended form of the dialog).

- Quick Navigation dialog **Class** combo box.

- Define Class dialog **Subclass of** combo box.

- Find Unused Class Entities dialog **Search selected class in selected schema** combo box.
- Find Unused Local Variables and Parameters dialog **Search selected class in selected schema** combo box.
- Find Possible Transient Class Leaks dialog **Search for creates of selected class in all schemas** combo box.
- Navigation bar in hierarchy browser combo boxes (that is, for classes, properties, and methods lists).

This feature uses standard combo box and list box facilities to achieve this filtering. You can achieve the same thing in your own JADE systems, as follows.

- Set the value of the **style** property for the **ComboBox** control class to **ComboBox.Style_DropDown** (0).
- The **ComboBox** or **ListBox** control class **hasPictures**, **hasTreeLines**, and **hasPlusMinus** properties are all set to **false**.
- The **sorted** property of the **ComboBox** or **ListBox** control class must be set to **false** after the list is loaded.
- A disabled **description** list item is added as the first entry in the list (after sorting is turned off).
- The **change** event on the combo box or associated text box calls a filtering routine to hide or show the entries based on the new text entered.

  Setting the level of the item to **2** hides the entries, which is why there has to be an entry at item position **1** that is always at level **1**. (A child item must have a parent.)

Note For a combo box, the **click** event is not normally implemented. The selection action is performed on the **closeup** event so that nothing happens until the user completes the selection action.

See also "Camel Case Filtering in List Boxes (NFS 67480, JAD-I-493)", elsewhere in this document.

Finding Unused Local Variables and Parameters (NFS 67629, JAD-I-507)

The **Find Unused Local Variables/Parameters** command, accessed from the Schema menu, now allows the filtering of unused parameters, as follows.

- The Parameters reporting options group box on the Find Unused Local Variables and Parameters dialog now contains the **Ignore Inherited Methods** check box.

  Check this check box if you want to exclude the parameters of any method that has reimplementations from the parameters unused list, regardless of whether the parameters are used.

  This covers the case where the superclass implementation of a method are essentially blank default implementations that are then properly defined in the subclasses.

- If you include the tag **[ExcludeFromUnusedParameterReport]** (which must be enclosed within **[ ]** bracket symbols) on the same line as an unused parameter, that parameter is not included in the report. For example, the first of the following code fragment comments does *not* exclude the unused parameter from the report, but the second example does.

  ```
  run(indx: Integer, str: String) updating; // [ExcludeFromUnusedParameterReport]

  run(indx: Integer, // [ExcludeFromUnusedParameterReport]
      str: String) updating;
  ```

  The new **UnusedParameterReport** global constant category contains the **"ExcludeFromUnusedParameterReport"** global constant, which you can use to locate the string tag in comments by using the local or global search functionality.

Going to a Specified Method Line Number (JAD-I-240)

You can now go to a specified line in a method displayed in the editor pane, including of the Sender Browser. (In earlier releases, you could go only to a specified code position.)

When you select the:

- New **Go To Line** command (Ctrl+Alt+G) from the Methods menu, the Find Position in Method Source dialog is then displayed.

  The **Line Number** option button is selected by default, the number of the line at which the caret is currently positioned is displayed, and the **Specify Line Number** text box displays the range of line numbers in the current method as hint text. Enter the number of the line you want to locate.

- Existing **Find Code Position** command from the Methods menu, the Find Position in Method Source dialog is then displayed.

  The **Code Position** option button is selected by default, the number of position at which the caret is currently positioned is displayed, and the **Specify Code Position** text box displays the range of code positions in the current method as hint text. Enter the number of the code position you want to locate.

When you specify a valid line number or code position and click **OK**, the caret is then positioned in the editor pane at the start of the specified line or at the specified position in the method.

Horizontal Scroll Bars in Hierarchy Browsers (NFS 67347)

All list boxes in hierarchy browsers now set the value of the **scrollBars** property to **Both** (**3**), so that a horizontal scroll bar is displayed only if any list entry is truncated horizontally.

Identifying User of Unavailable Resource (NFS 67278, JAD-I-303)

When the JADE development environment outputs a message stating that an unavailable resource (for example, a form, class, application, lock, and so on) is held by another user, the message now includes ***user-name {process-identifier}***. This enables you to determine which process to force off, and it minimizes errors when doing so.

The following is an example of the additional information output with the message.

```
    The Form ViewSchema::UpdateCustomer is in use by Wilbur {6}
```

Importing and Exporting Browser Layouts (NFS 67232, JAD-I-396)

Hierarchy browser form layouts that you created following the release of JADE 2018.0.01 can now be saved.

Layouts assigned to a specific schema and the default layout for all schemas can now be saved (by selecting the **Export Preferences** button on the **Miscellaneous** sheet of the Preferences dialog) to your JADE user preferences initialization file, which defaults to **JadeUserPreferences.ini**. Conversely, the **Import Preferences** button enables you to apply a saved preferences initialization file.

Loading a saved user preferences initialization file (by selecting the **Import Preferences** button on the **Miscellaneous** sheet of the Preferences dialog) loads any saved layouts if the schema of the layout is present or if the layout is the default.

Note Loaded layouts replace any defined layout for the same schema or for the default layout.

Layouts for a schema that is not present are ignored. Existing layouts in your system are retained when there is no corresponding layout in your user preferences initialization file.

Loaded layouts are installed (applied) only when you click the **OK** button on the Preferences dialog after the preferences initialization file has been successfully read.

Loaded layouts are not applied to any open window or form. You must select the layout that your require from the View menu **Layout** command submenu.

Interfaces

This section describes the JADE interfaces changes in this release.

Interface Deletion (PAR 66088)

When you delete an interface, you are now informed that:

- You cannot delete an interface that is part of an exported package.

  You must remove the interface from the package before you delete the interface.

- An imported package that you are deleting contains an interface that the schema then extends. A Confirm Delete message box is displayed.

  If this extended interface is also used (implemented), an error message box informs you that you cannot delete the package, and you must first remove any implementations before you can delete the package.

Interface Implementation Mapper Dialog (NFS 48016, JAD-I-350)

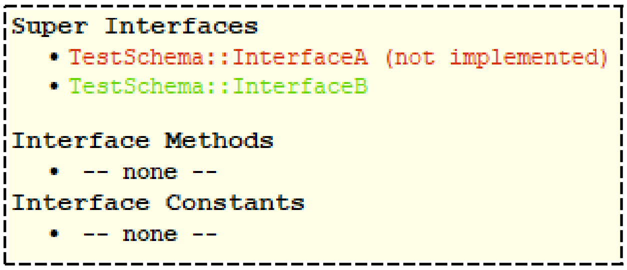
The Interface Implementation Mapper dialog has been updated to incorporate some new features based on NFS and JEDI suggestions.

You can now change the selected class using the new **Selected Class** drop-down combo box at the top of the Interface Implementation Mapper dialog, which enables you to quickly transition to different classes without closing and reopening the dialog every time.

Available and implemented interfaces have been separated into two list boxes, making it easy to distinguish between interfaces already implemented and those that are available to be implemented. In addition, a new **Show Inherited** check box is displayed above the implemented interfaces list box, which when checked, displays the interfaces implemented on any superclasses of the currently selected interface in the interfaces list box.

The inherited interfaces also display the schema and superclass in which they are implemented.

The interface summary text box has been updated to include information about any super interfaces belonging to the selected interface. If the selected class does not implement one or more of these prerequisite super interfaces, they are displayed in red with the expression (not implemented) beside their name. In addition, if a super interface implementations is missing, the **Implement** button is disabled. If the super interfaces are implemented or inherited by the selected class, they are displayed in green; for example:


```
Super Interfaces
   • TestSchema::InterfaceA (not implemented)
   • TestSchema::InterfaceB

Interface Methods
   •  -- none --
Interface Constants
   •  -- none --
```

The method mapping table has been updated to better display compatible methods. If an existing compatible method is selected for mapping, it is displayed in bold. In addition, a new **Map To Existing Methods** check box is displayed below the method mapping table. When this is checked, any existing method with the same name as a corresponding interface method name and with a compatible signature is automatically selected for mapping. This check box is checked (enabled) by default and overrides the value of the **Include Method Prefix** check box.

Interfaces Displayed for a Class (NFS 66482)

When a class is selected in the Class List of the Hierarchy Browser, the editor pane now displays any inherited interfaces within the schema and superschemas and any interfaces implemented in subschemas.

If the class does not implement any interfaces, none of the following sections are displayed.

- Implements the following interfaces:

  schema-name::*interface-name*

- Inherits the following interfaces:

  schema-name::*interface-name* from *schema-name*::*class-name*

- Subschema implemented interfaces:

  schema-name::*class-name* implements *schema-name*::*interface-name*

- Superschema implemented interfaces:

  schema-name::*class-name* implements *schema-name*::*interface-name*

Iterating using Interfaces (JAD-I-571)

The **foreach** instruction has been extended to support iterating with the **JadeIterableIF** interface, which provides the contract for an implementing class to be able to be iterated. **JadeIterableIF** simply exposes a **JadeIteratorIF** implementation through its **createIterator** method. This exposed **JadeIteratorIF** implementation can then be used to iterate the **JadeIterableIF** receiver.

A **JadeIterableIF** interface instance, or similarly a **JadeIterableIF** implementor instance, can be the target of a **foreach** instruction. For details, see "Iterating using the JadeIterableIF Interface", in Chapter 1 of the *JADE Developer's Reference*.

The **JadeIterableIF** interface is implemented by the RootSchema **Collection** class, which satisfies the **createIterator** method by creating RootSchema **Iterator** class instances.

For details about implementing the **JadeIterableIF** interface for a class selected in the Class Browser of a user schema, see "Implementing an Interface", in Chapter 14 (Adding and Maintaining Interfaces"), of the *JADE Development Environment User’s Guide*. See also "Processing and Obtaining Information about Dynamic Objects", elsewhere in this document.

Mapping Interfaces (NFS 67750, JAD-I-273)

If you use Shift+F11 to navigate through a series of methods and you reach a method defined on an interface, there was no quick way to get a list of the classes and methods that implement that interface.

The handling of the display of interface mappings has now been enhanced, as follows.

- Selecting the **Interface Mappings** menu command in the Methods menu of a hierarchy browser displays all interface methods that the method implements and the entire list of methods that implement each interface method.

  This shows each interface method at level 1 in the list box and all methods that implement the interface underneath at level 2 (sorted by name) with drawn tree levels and icons; for example:

```
RootSchema::JadeMultiWorkerTcpTransportIF::connectionEvent (interface method
and the following methods that implement it)
     RootSchema::JadeOdbcServer::mwtt_ConnectionEvent
     RootSchema::JadeWebAppExtension::mwtt_connectionEvent
```

- Selecting the **Method Implemented by** command in the Methods menu from the Interfaces Browser displays the same information as that in the previous item of this list.

- The Methods menu commands for all Method windows now also include the **Interface Mappings** command. (This includes all Shift+F11 method lists.)

  This command is enabled only when the currently selected method implements an interface or it is a an interface method.

Method List and Interface Folder Changes

Methods that have one or more interface mappings displayed in the Methods List window in the Class Browser now display **(i)** on the end of the method name; for example:

- addCustomer (i)

Note If the method is also a reimplementation, it displays both **(r)** and **(i)** on the end of the list item method name.

To view all implemented interface mappings, select the **Interface** folder in the Methods List window in the hierarchy Class Browser. The Methods list then displays all implemented interfaces and mapped methods as a two-level hierarchy. The top level displays the names of all interfaces implemented on the selected class and the second level displays all their implemented methods. If a method has multiple interface mappings, it displays the method under each of the mapped interfaces. If the implemented method name differs from the interface method name, the interface method name is displayed in parentheses after the implemented methods name.

To group implemented methods into interfaces in which they are defined for all Methods List folders, select the **Show Interface Method Mappings** command from the View menu.

Note Selecting the **Show Interface Method Mappings** command from the View menu does not change what is displayed when viewing the **Interface** method list folder, as it already displays this information.

Typecasting Interface Objects (NFS 63986)

For convenience and to help safeguard from typecasting to a class that does not implement the interface at compile time, JADE now allows you to typecast interface objects directly to any class implementing the interface without first having to typecast to the **Object** class; for example:

```
football := iSports.Football;
```

In earlier releases, it was not obvious that you had to first cast to the **Object** class; for example:

```
football := iSports.Object.Football;
```

Loading Checked Out Methods (JAD-I-181)

When a delta is set, the **Check Out Methods** check box on the Advanced Load Options dialog is now checked by default. (In earlier releases, it was unchecked.) This was done to reduce the chance of overriding existing methods.

If a delta is set and it contains methods checked out prior to a schema load, they are updated with any change in the schema file. If you want to preserve your current system, set a delta with no methods checked out prior to a schema load, and make sure the **Check Out Methods** check box is checked (it is checked by default if a delta is set).

Opening a Method from the Method Status List Browser (NFS 66744)

When you double-click on a method displayed in the methods list of the Method Status List Browser or you double-click a method entry while holding down the Shift key, that method is then displayed in a separate window.

This enables you to fix a compile error and keep the method displayed. (When you fix a compile error in the editor pane of the Method Status List Browser and press F8, the method is removed from the list so is no longer displayed.)

Orphaned Event Method Detection (NFS 67029)

When a control or menu is deleted from a form, in the past, not all associated event methods of the control or menu were always deleted, which can cause schema load failures because the control is unknown. Such orphaned methods are marked as events methods and do not appear on the **All** sheet of the Methods List in the Class Browser but they *are* displayed in the Methods Browser for a class.

These orphaned event methods are now detected when you use the Validate Schema dialog to validate forms. Orphaned event methods are displayed in the error output, as follows.

- Form *form-name* has an event method *method-name* that does not have an associated property

For form event methods without an underscore character (_), the following error is output.

- Form *form-name* has an event method *method-name* that does not have an associated property and is not a **Form** event
- Form *form-name* has an event method *method-name* that does not have an associated property but a non-control or menu property also exists: *property-name*

Painter

This section describes the JADE Painter changes in this release. (See also "Skin Category Selection in the JADE Painter (NFS 67295)" under "Skins", elsewhere in this document.)

Deleting the Current Form in Painter (NFS 67602, JAD-I-520)

The File menu in the JADE Painter now provides the **Delete Current Form** command, which enables the currently active form to be deleted while allowing you to visually confirm that it is the correct form.

If the form can be deleted (that is, it has no meta data inhibitors such as has property references), the menu command is enabled and the caption is **Delete Current Form:*active-form-name***.

If the form cannot be deleted, the menu command is disabled. Placing the cursor over the disabled command displays in the status line the reason that the form cannot be deleted (for example, the form has subclasses).

Clicking the **Delete Current Form** command:

- Reconfirms that the form can be deleted.
- Checks that the form is not in use elsewhere in the system.
- Confirms that you can perform the delete action (that is, the security library is called).
- Deletes the form and its class if there are no inhibitors.
- Closes the form in Painter if the form is deleted and discards any unsaved changes.

Finding Available and Duplicate Caption Accelerator Characters (NFS 67617)

In earlier releases, the Painter Menu Design dialog enabled you to see the available characters in a menu caption.

The Controls menu in the JADE Painter now provides the **Find Available Caption Accelerator Characters** and **Check For Duplicate Control Accelerators** commands, which show the characters in a control caption that are available to be used as a unique accelerator and the duplicate control accelerators in the full hierarchy of the currently selected form, respectively. These Controls menu commands result in the display of a message box containing:

- All of the used accelerator characters for other controls on the form, all superclasses, and all subclasses of the current form gathered and removed from the caption string of the currently selected control, leaving the list of available characters

- All of the accelerators on all forms in the hierarchy of the form are searched for duplicates

After selecting a control, clicking the **Find Available Caption Accelerator Characters** command, the Accelerator Search message box displays one of the following.

- There is no character in the caption that can be used as a unique accelerator
- The character *character* is available to be used as an accelerator
- The following characters are available to be used as an accelerator: *character-list*

The **Find Available Caption Accelerator Characters** command is disabled if a control is not selected or if the selected control does not have a **caption** property.

After selecting the **Check For Duplicate Control Accelerators** command, the Accelerator Duplicate Check message box displays one of the following.

- There are no duplicated accelerators on any control in the hierarchy for the form *selected-form-name* [including sub-classes]
- There is a duplicate use of a control accelerator as follows: '*accelerator-character*' [*form-name*::]*control-name* [*form-name*::]*control-name*
- There are duplicate uses of control accelerators as follows: *'accelerator-character*' [*form-name*::]*control-name* [*form-name*::]*control-nameaccelerator-list*

The **Check For Duplicate Control Accelerators** command is disabled if a form is selected or if the selected control does not have a caption property.

Positioning Controls and all Parents in Painter (NFS 67670)

The Painter Controls menu now enables you to bring a control and all of its parents to the top of the zOrder of controls displayed on the form or to push them all to the bottom of the zOrder.

Printer Form Font (PAR 67048)

For printer-style forms in JADE 7.1 and earlier releases, controls were given the default font Arial 10. In JADE 2016, controls added to printer-style forms are given the default font Tahoma 10, and all new printer forms or all new controls added to printer forms default to Tahoma. This default is hard-coded and cannot be changed.

From version 2018.0.02, JADE has been changed to provide you with the ability to define default property values for new controls added to a printer form in the JADE Painter.

The following new methods are available.

- RootSchema **Window** class **setDefaultPainterControlProperties** method
- RootSchema **Form** class **isPrinterForm**

The **setDefaultPainterControlProperties** method does nothing, by default. The JADE Painter calls this method on any new control created in the painter. You can re-implement this method and set default property values to meet your requirements.

The base method is declared in the **Window** class so that you can re-implement the method in the **Control** class in your schema to set any control properties. You can also re-implement the method on any **Control** subclass, to set properties on a specific control subclass; for example, **Frame**.

The **Form** class **isPrinterForm** method returns whether the form was declared as a printer form on the New Form dialog in the JADE Painter; that is, the **Printer** option button was selected in the Form Style group box. This method then allows the **setDefaultPainterControlProperties** method re-implementation to set properties only on a printer-style form, for example.

The following method is an example of these methods.

```
setDefaultPainterControlProperties();
begin
    if self.form.isPrinterForm() then
        self.fontName := "Arial";
    endif;
end;
```

For more details about printer-style forms in the JADE Painter, see "Printer-style Forms", in Chapter 5 of the *JADE Development Environment User's Guide*.

Removing ActiveX Controls from Forms (NFS 67595, JAD-I-530)

When an ActiveX control cannot be located and installed, you cannot run or edit the form on which it resides.

If an ActiveX control cannot be loaded when a form is opened for editing, the JADE Painter now displays a message box, prompting you to confirm that the load should continue. The message box displays:

```
The load of the control control-name of type control-type failed with an exception:
error-text error-code
```

Click:

- **Yes** to continue the load action.

  The loading of the form continues and the control is marked as being deleted. The control and any event methods for the control are permanently deleted from the form when you save it.

- **No** to abort the load action.

  The form load is aborted and abandoned.

Note The form load cannot be continued if the control being loaded is on a superform or if the control would have had children.

The ability to delete controls from a command file and from the JADE development environment Class Browser will be considered in a future release. The only current work around is to extract the class and manually remove the control from the ***.cls** and ***.ddb** or ***.ddx** files.

Title Bar (NFS 67198, JAD-I-309)

The Painter title bar now includes the database path, user identifier and process identifier, and the skin (if selected) in the caption. The information is displayed in the following format.

```
JADE Painter : (database-path : user-identifier_painter-process-identifier :
singleUser or multiUser) schema-name::form-name skin-selected
```

Pasting Text into the Method Definition Dialog (NFS 67346)

You can now paste text into the **Name** text box of the Method Definition dialog so that the paste action displays the initial name in the text box as the name of the new method.

The paste action strips all but the first identifier (alpha, numeric, and underscore characters) and inserts that text into the **Name** text box. This enables a simpler selection process when you copy the text of an existing method (for example, when you have pressed Ctrl+A followed by Ctrl+C to copy the entire method text to the clipboard followed by Ctrl+V to paste it into the **Name** text box on the Method Definition dialog).

For example, pasting **get_Customer_Details(formatType : Integer): String;** in the text box results in the following displayed as the method name.

```
get_Customer_Details
```

Patch Versioning

This section describes the patch versioning changes in this release.

Comparing Method Sources (NFS 67704, JAD-I-160)

When comparing method source within a patch summary, the Compare Sources dialog now hides the merge editor pane by default, which provides more space for the displayed method sources.

A **Show Merge Pane** button is now provided, which when clicked, toggles the visible status of the merge editor pane. The button has the **Show Merge Pane** caption when the pane is hidden and the **Hide Merge Pane** caption when the pane is visible.

Notes The size of the merge editor pane is no longer saved and restored.

When the merge editor pane is visible, from JADE 2016, a resize bar enables you to drag the merge editor pane to reduce or increase its displayed size in the dialog.

Primitive Types Browser Type Menu (PAR 66989)

When a Primitive Types Browser is displayed, clicking on the **Methods List** folder did not correctly set up the editor display for the sheet **change** event method so that the previous method was displayed in read-only mode.

The behavior of the Class Browser and Primitive Types Browser has now been changed to fix this issue. When you now select another sheet in the **Methods List** folder, the first defined method in the list is selected in the methods list and is automatically displayed in the editor pane, ready for editing.

If there is no defined method in the list, the editor pane displays the information for the currently selected class or primitive type.

Refactoring JADE Methods

This section describes the changes to the refactoring of JADE methods in this release.

Identifying Local Variables in Extracted Methods (NFS 67432)

JADE now detects local variables used only in the selected logic of methods to be extracted. When the refactored method is created, they are now defined as local variables in the new method rather than being passed as **io** parameters in the signature of the created method. The definition of local variables is not affected in the original method.

If a variable is used in other logic not selected in the original method to be extracted, it is still passed as a parameter to the new method that is created.

Signature of Created Methods (NFS 67428, JAD-I-498)

In earlier releases, when creating a new method using the **Extract Method** command from the editor pane popup (context) menu **Refactor** command, a public method was always created.

When you extract a method for refactoring, a protected method is now created if you have selected the **Protected** access option button in the Methods group box on the **Schema** sheet of the Preferences dialog. In the editor pane, manually remove the **protected** option from the signature of the created method if it is not required.

Running an Application from the String Browser (NFS 67291, JAD-I-272)

The File menu in the String Browser now contains the **Run Application** (Ctrl+R) command, which enables you to run an application from the form on which you translate strings.

Scaling Decimal Array Elements (NFS 64143)

If you select the **Decimal** primitive type for membership of an array class, the **Scale Factor** text box is now enabled on the **Membership** sheet of the Define Class dialog. To maintain compatibility for existing applications, **Decimal** values added to an array of decimals are not scaled, by default.

When you check this check box to define the scale factor for entries in the array, values are rounded to the number of decimal places specified by the scale factor when they are added to the array.

Schema Navigation (NFS 67352, JAD-I-484)

Changing the schema using the navigation bar combo box in the Class List of a Class Browser to navigate to the parent class in a superschema no longer selects the **Object** class in the superschema by default.

The selected class is now the first class previously selected in the class hierarchy that is common to both schemas; that is, if a:

- Superschema is selected, the class selected will be the superclass of the previously selected class.
- Subschema is selected, the previously selected class will again be selected in that schema.

Searching

This section describes the search functionality changes in this release.

Exposure Browser Search (NFS 67226, JAD-I-334)

The Exposure Browser now provides the ability to search for an exposure by name, by right-clicking in the list box and then selecting the **Find** command. The Find Exposure dialog is then displayed, listing the current exposures in alphabetic order.

The list is filtered when you specify text in the **Find** text box, so that the displayed exposures are only those entries that contain the specified text. The text is case-insensitive or if the specified text is all uppercase, the entries with the same Pascal case are located; for example, **DCE** matches **DocCsharpExample**.

If all text entered after the first character is uppercase, the list 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.

Select an entry in the **Select Required Entry** list and then click the **OK** button so that the exposure is selected in the Exposure Browser.

Finding a Class by Number (NFS 67605, JAD-I-574)

The Find Type dialog now enables you to specify a class number to search for the associated class in the current schema so that specifying numeric values does not affect the search.

The Find Type dialog has changed as follows.

- When initiated from a hierarchy browser, the dialog now has a **Find Class Number** check box, which is unchecked by default.
- If the check box is unchecked, specifying text in the **Find** text box filters the list of displayed class names as in earlier releases, so that it includes the situation where you specify numbers only.
- If the check box is checked, the **Find** text box is cleared and allows only numeric digits to be specified.

  Pressing Enter searches the current schema and all superschemas for that class number. If that class number:

  - Is found, the hierarchy browser is refreshed to select and display that class.
  - Cannot be found, a message box is displayed.

- The check box is not visible when performing a search in other hierarchy browser forms such as the Schema Browser and the Global Constants Browser.

Global Searching for an Imported Entity (PAR 66489)

The global find and replace functionality now enables you to search for and optionally replace an entity (for example, a method) imported from another schema.

If you want to search the names and values of all imported entities that match your search criteria, check the **Include Imported** check box in the Search Entities group box on the Global Search And Replace dialog. (This check box is unchecked by default.)

Saving and Restoring Find and Replace Options (NFS 67614, JAD-I-255)

The local Find/Replace dialog and the Global Search and Replace dialog have been changed so that the **Case Sensitive** and the **Full Word Only** check box values are saved, and restored the next time the dialog is displayed.

The first time the dialog is displayed after logging on to the JADE development environment, the values are **false** (that is, unchecked). After you perform a search, the values used in the search are saved. Those values are restored the next time the dialog is used in your current development environment session.

Searching for a Property or Method in a List (NFS 67674, JAD-I-550)

In earlier releases, the Ctrl+6 and Ctrl+7 shortcut keys enabled you to search a hierarchy Class Browser for a specific property or method, respectively, in the hierarchy Class Browser. The displayed list of properties or methods is filtered based on what you type into the search text box.

These shortcut keys now enable you to search all properties or methods displayed in a list box in the:

- Class Browser
- Class References Browser
- Messages Browser
- Methods Browser
- Interface Browser
- Primitive Types Browser
- Sender Browser lists such as references and implementors
- Unused Entities Browser

Searching for Changed Methods by Date and Time (NFS 67718, JAD-I-409)

The Changed Methods dialog that is used to search for changed, checked out, and versioned methods now provides the **From Time** and **To Time** text boxes, which enable you to filter located methods based on the date and the time. These text boxes are associated with the *from* and *to* dates. If the corresponding date is not specified, the time value is ignored.

The values of these text boxes default to **00:00** and **23:59**, respectively. The *to* time includes any part of the specified minute.

When you specify a date and time, the search includes any methods that were changed after the specified from date and time (that is, the timestamp) and before the specified *to* date and time.

If the *to* date is not specified, the search includes methods changed after (or at) the specified *from* date and time.

Searching for Comments (NFS 67628, JAD-I-534, PAR 67891)

The local Find/Replace dialog and the Global Search and Replace dialog now provide the **Include Comments** check box, which is checked by default, applies only to a find or replace in a JADE method. By default, comments in the source of a method are searched for and optionally replaced.

Tip You can use the **Alt+u** shortcut keys to toggle the check box value.

When the check box is unchecked, occurrences of the search text in method comments, including line (**//**) and block (**/* */**) comments, in the source of a method are ignored when searching and replacing.

The **Include Comments** check box on the Global Search and Replace dialog is not displayed if the **Methods** check box is not checked, as it does not apply to the search of class or global constants.

The **Include Comments** check box is set to **true** (that is, checked) for the first search performed after logging on. Subsequent displays of the local Find/Replace and the Global Search and Replace dialogs set the check box value to the prior setting that was used to perform a search or replace action.

Sorting Possible Transient Leaks (NFS 67298, JAD-I-474)

The possible transient leaks listed in the upper pane of the Methods Browser are now sorted in ascending order of the **Method** column (that is, by schema, class, and method).

Left-click on the first row (the column header) of the table to sort the possible transient leaks in descending order (that is, **Z** through **A** or **9** through **1**) using the contents of the clicked column. Click the same column again to toggle the order of the entries.

The column that is used for sorting displays an arrow icon to indicate whether the sorting is in ascending (^) or descending (v) order.

Suppressing Message Boxes (JAD-I-100)

There are a number of messages boxes that appear frequently to which the response is usually always the same, for example:

- After a schema load: *Load Completed* or a **.mul** file load of multiple schemas: *Load Completed Successfully*
- When a change prompts recompilation: *Your change requires n method(s) to be recompiled. Recompile now?*
- When compiling a method: *Do you want to view the error list?*
- Confirmation when moving or copying a method
- When adding a method: *Do you want to reimplement superclass method?*
- Find and replace in the editor pane: *Search text not found* and *Replace successful*

You can now specify that the display of a number of message boxes is suppressed and your answers are retained for your future work sessions. When a message box is suppressed, the button you set is recalled and its **click** event is automated.

By default, no message boxes are suppressed. Suppress message boxes by performing one of the following actions.

- When a message box that can be suppressed is displayed, check or uncheck the **Hide in the future** check box if there is one button only. If there is more than one button, select the button you want to be automated and then check the **Remember my selection and hide in the future** check box.
- On the Preferences dialog, select the new **Message Box Suppression** sheet and then enable or disable each message box in the displayed table.

The **Message Box Suppression** sheet of the Preferences dialog enables you to check the suppression of each message box in the **Suppressed** column and then in the **Stored Value** column, select the button in the combo box that is to be automatically clicked when that message box is displayed.

Note Your suppression options are not committed until you click the **OK** button on the Preferences dialog.

When the **Stored Value** embedded combo box is set to the first empty entry in the combo box, the **Suppressed** check box is unchecked. Conversely, if the **Suppressed** check box is not checked and you select a valid stored value in the embedded combo box, the **Suppressed** check box is checked.

Click the **Suppress All** button to suppress all message boxes listed in the table, or the **Clear All** button to clear all of your message box suppression settings.

The **Pop-up Display Time (ms)** text box enables you to specify the number of milliseconds that the Status Line Popup dialog is displayed in the lower-left corner of the JADE development environment if you want to decrease or increase the default display time of 4.75 seconds (that is, 4750 milliseconds). The default display time can be within the range 1500 through 15000 milliseconds, which adequately updates the user to the system state while suppressing the message-box and requiring no user-interactions. If you set a value lower than 1500 milliseconds or higher than 15000 milliseconds, the minimum or maximum value is set after a message box indicates that the value is invalid.

The Status Line Popup dialog is displayed indefinitely when the cursor is over it. The timer resumes after the cursor ceases to hover over the dialog. Press Esc or click the close button, marked with **X**, to close the Status Line Popup dialog immediately. Check the **Hide Pop-up** check box if you do not want the Status Line Popup dialog displayed. By default, this check box is unchecked.

Viewing Defined References (NFS 66564)

The **Defined Inverses** pane on the extended Define Reference dialog has been moved to a pane that is above the buttons at the foot of the dialog, and which spans the width of the dialog so that it is easier to view inverse references without having to resize the dialog. In addition:

- The relationship cardinality between the current and related classes has been updated.
- A status bar has been added to the bottom of the dialog, to display the effect on the relationship between the two properties when you click an access, update mode, or relationship type option button.
- Check boxes and access, update mode, and relationship type option buttons now display bubble text indicating their function when you hover over the control.

JadeAuditAccess Read Offset (PAR 66198)

Because the **JadeAuditAccess** read offset can be incorrect if large negative or large positive **Integer** values are used, the following methods are now defined in the **JadeAuditAccess** class.

-
  ```
  getJournal_64(pDirectory:    String;
                pJournalNumber: Integer64;
                pRecordOffset:  Integer64 io): Integer updating;
  ```

-
  ```
  getJournalNumber_64(): Integer;
  ```

-
  ```
  getNextRecordUTC_64(pType:         Integer output;
                      pObjectType:    Integer output;
                      pRecordOffset:  Integer64 output;
                      pUTCTimestamp:  TimeStamp output;
                      pUTCBias:       Integer output;
                      pTimestamp:     TimeStamp output;
                      pSerialNumber:  Decimal output;
                      pTransactionId: Decimal output;
                      pOid:           String output;
                      pClassNumber:   Integer output;
                      pEdition:       Integer64 output): Boolean;
  ```

-
  ```
  getNextRecord_64(pType:         Integer output;
                   pObjectType:    Integer output;
                   pRecordOffset:  Integer64 output;
                   pTimestamp:     TimeStamp output;
                   pSerialNumber:  Decimal output;
                   pTransactionId: Decimal output;
                   pOid:           String output;
                   pClassNumber:   Integer output;
                   pEdition:       Integer64 output): Boolean;
  ```

The **JadeAuditAccess** class **getJournalNumber**, **getNextRecordUTC**, and **getNextRecord** methods can now return exception 1406 (*Result of expression overflows Integer precision*) or 1446 (*Result of expression underflowed Integer precision*) if the offset or journal number parameters exceed **Max_Integer** (2,147,483,647) or go below zero (**0**). If this occurs, you must use the corresponding **_64** version of the method.

JADE Initialization File

This section describes the JADE initialization file changes in this release.

AccessibilityEnabled Parameter (NFS 67762)

The [Jade] section of the JADE initialization file can now contain the **AccessibilityEnabled** parameter, which controls whether JADE processes accessibility messages. (This parameter is read when the client node is initialized.)

The parameter is set to **false** by default, as this feature is not required by most users. The default value means that JADE will not participate in the accessibility processing and Microsoft Windows uses its default handling for any accessibility features that are enabled on the client node.

Set the parameter to **true** if you want JADE to participate in accessibility handling. This mode then provides accessibility with additional information when performing blind reading, for example.

Single User Application Restrictions (PAR 65360)

Application restrictions are now applied for a node running in single user mode. This includes single-user **jade.exe** or application server. Only permitted applications can be started if the value of the **EnableAppRestrictions** parameter in the [JadeClient] section of the JADE initialization file is set to **true**.

Thin Client Security (PAR 66521)

In earlier releases, if the **EnableAppRestrictions** parameter in the [JadeAppServer] section of the JADE initialization was set to:

- **true** and another initialization file parameter enabled access to the JADE development environment (for example, **AllowSchemaAndApp1=JadeSchema,Jade** or just **JadeSchema**), any application that was run from RootSchema was also able to be executed.
- **false**, there were no restrictions on what application a thin client could run, which potentially allowed users to run applications that may not have been in the best interests of the site.

As a result, the security for running applications from a JADE thin client has now changed.

There has been a fundamental change to the **EnableAppRestrictions** parameter security checking. If the application was initiated by logic from another application, the application is now allowed to run. (In earlier releases, all thin client applications that were initiated were subjected to the **EnableAppRestrictions** parameter processing, which meant that any applications allowed by the **EnableAppRestrictions** parameter could be initiated by any thin client user.)

This new philosophy means that your organization can write your own applications that then initiate protected applications. Each application can then enforce its own **getAndValidateUser** sign-on security before initiating the protected application.

The **EnableAppRestrictions** mechanism enables you to prevent thin client users from directly executing a protected application.

Note Any application can be initiated from the JADE development environment unless prevented from doing so by the JADE development environment security mechanisms.

The [JadeAppServer] section of the JADE initialization file can now contain the Boolean **EnableRootSchemaAppRestrictions**=<default> parameter, which controls whether a user can execute RootSchema applications, in conjunction with the **EnableAppRestrictions** parameter.

The following **EnableAppRestrictions** and **EnableRootSchemaAppRestrictions** parameter combinations apply to the initial application initiated by a user. ('Child' applications initiated by that application are always allowed to run.)

EnableAppRestrictions	EnableRootSchemaAppRestrictions	Thin client user...
false	false	No restrictions on the applications that can be executed.
false	true	Cannot execute any RootSchema application but can run any other application.
true	true	Can execute only applications specified using **AllowSchemaAndApp<*n*> = <*schema*>,<*application*>** parameters. Including **AllowSchemaAndApp<*n*> = JadeSchema,Jade** or **= JadeSchema** does not grant the ability to execute RootSchema applications. Any RootSchema applications that are allowed must have their own **AllowSchemaAndApp<*n*>** parameters.
true	false	Can execute only applications specified using **AllowSchemaAndApp<*n*> = <*schema*>,<*application*>** parameters. Including **AllowSchemaAndApp<*n*> = JadeSchema,Jade** or **= JadeSchema** grants the ability to execute RootSchema applications.

The default value of the **EnableRootSchemaAppRestrictions** parameter is **true**, which means that RootSchema applications cannot be executed by default unless they are initiated by another application such as the JADE development environment.

JadeHTTPConnection Class

This section describes the **JadeHTTPConnection** class changes in this release. (See also "REST Service Security (JAD-I-430)", elsewhere in this document.)

isStatusCodeSuccess Method (JAD-I-97)

The **JadeHTTPConnection** class now provides the **isStatusCodeSuccess** type method, which returns **true** if the HTTP connection status code is in the range of successful responses specified by the new **Minimum_ Successful_StatusCode** and **Maximum_Successful_StatusCode** class constants; otherwise it returns **false**. (For details about HTTP connection status codes, see https://developer.mozilla.org/en-US/docs/Web/HTTP/Status.)

JadeHTTPConnection Class Supported Verbs (NFS 64197)

As the **JadeHTTPConnection** class now supports all HTTP verbs as well as the existing **"GET"** (the default value) and **"POST"** verbs, you can now specify other verbs such as **"DELETE"**, **"PATCH"**, and **"PUT"** in the:

- **pVerb** parameter of the **getHttpPage** and **getHttpPageBinary** methods
- **verb** parameter of the **sendRequest** and **sendRequestUtf8** methods

JadeHTTPConnection::sendRequestUtf8 Method (NFS 66868, JAD-I-349)

The **JadeHTTPConnection** class now provides the **sendRequestUtf8** method, which has the following signature.

```
sendRequestUtf8(verb:                String,
                additionalHeaders:   String;
                optionalPostPutData: StringUtf8): Boolean;
```

This method sends the specified request from an ANSI system as **StringUtf8** data to the HTTP server.

Note This method is the same as the existing **sendRequest** method except that the **StringUtf8** value of the third parameter (that is, **optionalPostPutData**) specifies a string, encoded in UTF8 format, containing any optional data to be sent immediately after the request headers.

JADE Monitor Cache Performance Details (JAD-I-515)

The **Cache Performance** view of the JADE Monitor now contains the **Details** check box, which you can use to toggle between the display of a summary of cache performance and cache performance details for each type of cache.

JadeTimeZone Class (JAD-I-254)

JADE now provides the **JadeTimeZone** class, which enables you to obtain information about and perform conversions between different time zones. It also supports differing daylight saving rules across different time zones.

JadeTimeZone objects are transient only. You cannot create persistent or shared transient instances.

Create your own transient **JadeTimeZone** subclasses to:

- Obtain information about time offsets and daylight saving for various regions; for example, coordinating communication between different time zones, providing information about time zones, having systems automatically perform actions switching over to or from daylight saving, and so on
- Use **JadeTimeZone** objects to perform timestamp conversions for different time zones and timestamps
- Convert a time zone for a past or future timestamp, where the daylight saving state may differ from the current daylight saving state

ListBox and Table Control Disabled foreColor (PAR 67920)

ListBox and **Table** controls now observe the **foreColorDisabled** property value if specifically set; otherwise the default system gray color is used.

You can override the default disabled color, as follows.

- If the **ListBox** or **Table** control uses a skin and that skin has a default disabled foreground color value set (from the **Default disabled foreColor** check box on the **Controls** sheet of the Jade Skin Maintenance dialog), that skin-defined color is used.

- If you programmatically set the **foreColor** property of a disabled **JadeTableCell**, this color overrides both the System disabled color (gray) and any skin colors.

Loading User Preferences from the Command Line (NFS 67599)

You can now load an exported user preferences file by executing a non-GUI JADE client application using the **jadclient** or **jade** executable; for example, if you have created and built JADE environments from a specific release and you want to load your preferences with a defined schema set.

The **LoadUserProfileIniFile** application requires the following two parameters.

- **userName**, which specifies the user name of the user profile to be created or updated

- **profileIniFileName**, which specifies the file path and name of the preferences **.ini** file that is to be loaded

The following is an example of the **jadclient** command line. (You can also load your user preferences by using the JADE executable (**jade.exe**) command line.)

```
jadclient.exe path=c:\Jade\system ini=c:\jade\system\jade.ini server=singleUser
schema=JadeSchema app=LoadUserProfileIniFile startAppParameters userName=myusername
profileIniFileName=c:\jade\JadeUserPreferences.ini
```

The exit of **jadclient** or **jade** is set to non-zero if the process fails. The success or failure is also written to the **jommsg log** file.

See also "Loading User Preferences from an Earlier Release (NFS 67599)" under "JADE 2020 Changes that May Affect Your Existing Systems", earlier in this document.

Logical Certifier

In addition to a number of new checks, this section describes the Logical Certifier changes in this release.

Additional Checks Resulting from Closed PARs

A number of Logical Certifier checks are now performed as a result of closure of the following Product Anomaly Reports (PARs).

- 66131 - Logical Certifier does not detect orphan DevControlProperties instances

- 66135 - Logical Certifier does not detect orphan MenuItem instances

- 67039 - Logical Certifier does not check Translatable Strings used in Controls, MenuItems and Forms

- 67285 - Logical Certify reports errors for exclusive collections defined on a subclass of JadeBytes

- 67690 - Logical Certifier does not detect orphan event methods

- 67813 - Logical Certifier does not detect MenuItem instances with no corresponding property

- 67817 - Logical Certifier does not detect Control instances with no corresponding property

- 67820 - Logical Certifier does not detect Property instances marked as control with no corresponding Control

The **Closure Details** sheet of Parsys provides information about the check or checks that have been incorporated.

Incremental Logical Certify (JAD-I-478)

The JADE Logical Certifier now provides an operation to validate only inverse references and collections that have changed since the last time the system was certified.

The Logical Certifier saves the highest update transaction ID of the inverses that are certified. On subsequent certifies, inverses are certified only if the update transaction ID is greater than the saved highest transaction ID. If the update transaction ID of an inverse is lower than the saved value, it must have been certified previously.

Checking only changed inverses incrementally can significantly reduce the time required to check the integrity of user data.

The Jade Logical Certifier dialog now provides the **Incremental Certify** option button for a full logical certify (that is, all classes in all schemas). When this option is selected, only inverse references and collections that have changed since the last time the system was certified will be validated. The Set Incremental Transaction ID dialog that is then displayed enables you to manually set the transaction ID.

The JADE Logical Certifier non-GUI application has new command line parameters to specify incremental certification and optionally to manually set the transaction ID used during an incremental certify.

MDI Child Forms

This section describes the Multiple-Document Interface (MDI) child form changes in this release.

Floating, Docking, and Pinning MDI Child Forms (NFS 67200)

You can now programmatically provide the functionality to float, dock, and pin MDI child forms in your own application logic. JADE now allows:

- Control over whether users can invoke an MDI menu on an MDI child form by right-clicking on the caption of the form. The MDI popup menu can contain the following commands.

Menu Command	Action
Close	Closes the form (the same as clicking the **Close** button or the Context-Menu **Close** command)
Close All But This	Closes all other MDI children in the current MDI frame except for the current form
Close All But Pinned	Closes all MDI child forms that are not pinned and have the **allowClose** property set to **true**
Float	Floats the current MDI child form
Dock	Re-docks the MDI child form in its MDI frame
Pin	Toggles the pinned status of an MDI child form

The **Form** class provides the following new **Boolean** primitive type properties, all of which have a default value of **false**, which means that the MDI menu is not displayed by right-clicking the MDI child caption:

Property	Contains the...
showMdiCloseAllButPinnedMenu	**Close All But Pinned** command
showMdiCloseAllButThisMenu	**Close All But This** command
showMdiCloseMenu	**Close** command

Property	Contains the...
showMdiDockMenu	**Dock** command
showMdiFloatMenu	**Float** command
showMdiPinMenu	**Pin** command

If the value of all of these properties is **true**, right-clicking the MDI child caption displays a menu.

These properties are ignored if the form is not an MDI child form. In addition, the **Close All But This**, **Close All But Pinned**, and **Close** commands are ignored if the **allowClose** property is set to **false**.

The property values can be set at run time and in the JADE Painter.

Note The **Float** command is disabled if the form is floating and the **Dock** command is disabled if the MDI child form is already docked in the MDI frame.

The **Form** class now also provides the **Boolean** type **mdiPinned** property, which defaults to **false**. This property is read and write at run time, to allow access from your application logic to the pinned status of a form.

- The **Form** class now provides the methods listed in the following table.

Method	Description
floatMdi();	Floats an MDI child. It does nothing if the form is already floating or if the form is not an MDI child.
dockMdi();	Docks an MDI child. It does nothing if the form is not floating or if the form is not an MDI child.
isMdiFloating(): Boolean;	Returns **true** if the MDI child is floating or **false** if it is docked or it is not an MDI child.

- The **Form** class now provides the event methods listed in the following table.

Event Method	Description
mdiFloated();	Called after the user floats an MDI child. Appears in the **Form Event Methods** list box in the JADE development environment. This event is not called if the **floatMdi** method is called to float the form.
mdiDocked();	Called after the user docks an MDI child. Appears in the **Form Event Methods** list box in the JADE development environment. This event is not called if the **dockMdi** method is called to dock the form.

Note When floated, an MDI child form is positioned on the screen in the same position, except it is not a child restricted to the MDI frame; for example, it can be dragged to another monitor.

The MDI child form is always on top of the MDI frame. To reposition the form programmatically, set the **left** and **top** properties, or use the **Window** class **move** method.

When an MDI child form is docked, the position and size of the form is restored to the values it had when it was floated, if the current top-most MDI child in the MDI frame is not maximized. If the current top-most MDI child in the MDI frame is maximized, the docked form is also maximized.

Tabs in MDI Child Forms (NFS 67224)

You can now programmatically provide the functionality to display tabs for MDI child forms in user systems, by controlling the style of MDI child form styles. The styles are:

- Standard MDI child forms, as in earlier releases (the default).
- Standard MDI child forms with a tab for each child form on the MDI frame. The child forms can be maximized, minimized, and restored.
- MDI with a tab only for each child form on the MDI frame. Only the top MDI child form is visible and it is always maximized.

For the second and third of the styles in the previous list, when tabs are displayed:

- The tab contains the caption of the child form. Clicking on the tab brings that form to the top. The child form with focus has its tab highlighted. If the caption is too long to fit in the tab, the first and last part of the caption are displayed separated by points of ellipsis (...).
- Moving the mouse over the tab displays the full caption in a bubble help display.
- A down arrow button is displayed at the end of the tabs. Clicking the button displays the full list of captions of open child forms.

  Selecting an entry in the list brings that form to the front (that is, the same functionality that the Window menu provides for arranging and manipulating child windows in the JADE development environment). The order of the list can be alphabetic, the last-used, or creation order.
- Not all tabs are displayed if there is not sufficient room. (Use the down arrow or Window menu to locate a form that is not displayed.)
- JADE provides the ability to pin selected tabs to the left of the displayed tabs. The user can also drag tabs to another position, by clicking the tab and dragging it. (It can be dragged only within the pinned or non-pinned grouping.)

  Pinned forms display a pin icon in the tab. Clicking the pin also unpins the form.

The **Application** class provides the following properties and class constants.

Note These properties apply only to forms that have been created in version 2020.0.01 and higher.

- The Integer type **mdiStyle** property, which defaults to **MdiStyle_Mdi** (0), sets the default MDI style for an application at run time. You can also set the application style in the Mdi Style group box on the **Form** sheet of the Define Application dialog in the JADE development environment.

  Set this property to one of the following values.

  - **MdiStyle_Mdi** (0)
  - **MdiStyle_Mdi_With_Tabs** (1)
  - **MdiStyle_Tabs_Only** (2)
- The Integer type **mdiWindowListOrder** property, which defaults to **MdiWindowList_Order_Creation** (0), sets the order that child forms are displayed in the Window menu list of child forms and in the MDI Tabs down arrow list. You can also set the application window list order in the Mdi Window List Order group box on the **Form** sheet of the Define Application dialog in the JADE development environment.

  Set this property to one of the following values.

  - **MdiWindowList_Order_Creation** (0)

    The Window List menu and the MDI tabs down arrow show the list of MDI child forms in creation order.

- **MdiWindowList_Order_LastUse** (1)

  The Window list menu and the MDI tabs down arrow show the list of MDI child forms in last-use order.

- **MdiWindowList_Order_Alphabetic** (2)

  The Window list menu and the MDI tabs down arrow show the list of MDI child forms in caption alphabetic order.

Message Boxes

This section describes the message box handling changes in this release. (See also "Suppressing Message Boxes (JAD-I-100)" under "JADE Development Environment", elsewhere in this document.)

Customizing Message Box Button Captions (NFS 66844, JAD-I-74)

JADE now provides the **Application** class **msgBoxCustom** method, which enables you to display a message box with customized button captions and waits for the user to click a button. The method displays a message box with as many buttons displayed as there are string captions specified in the **btnCaptions** parameter.

The new method has the following signature.

```
msgBoxCustom(msg:         String;
             title:       String;
             flags:       Integer;
             btnCaptions: ParamListType): Integer;
```

Note You can call this method only in a GUI application, because it is implemented in **jade.exe**.

The method parameters are listed in the following table.

Parameter	Specifies the...
msg	Message displayed in the dialog.
title	Dialog title (null (**""**) displays the application name).
flags	Icons that are to be displayed in the dialog, the cancel button, and the default button.
btnCaptions	String captions for the buttons to be displayed. The number of captions parameters defines the number of buttons to be displayed (in the range **1** through **5**).

The code fragment in the following example shows a **msgBoxCustom** method call.

```
retVal := app.msgBoxCustom("Please choose how many of the articles you require",
"Selection", MsgBoxCustom_Icon_Question_Mark + MsgBoxCustom_Default_First +
MsgBoxCustom_Cancel_Four, "1", "2", "3", "None");
```

The return value is the number of the button clicked by the user. The default button has been set to the first button and the fourth button is the button that will be clicked when the Esc key is pressed.

The new **MessageBoxCustom** category provides global constants for this method.

Caution Do *not* use the global constants in the **MessageBox** category, because some **MessageBoxCustom** global constants are synonyms while others do not apply (for example, buttons required).

The **flags** parameter is constructed by adding a constant from each of the first three groups. Use one constant only from each group. The global constants in the **MessageBoxCustom** category are listed in the following table.

Global Constant	Integer Value	Description
MsgBoxCustom_Cancel_None	0	There is no **Cancel** button (the default), and pressing Esc will be ignored
MsgBoxCustom_Cancel_One	1	First button is the **Cancel** button
MsgBoxCustom_Cancel_Two	2	Second button is the **Cancel** button
MsgBoxCustom_Cancel_Three	3	Third button is the **Cancel** button
MsgBoxCustom_Cancel_Four	4	Fourth button is the **Cancel** button
MsgBoxCustom_Cancel_Five	5	Fifth button is the **Cancel** button
MsgBoxCustom_Default_First	MsgBox_Default_First (0)	First button is the default button (default)
MsgBoxCustom_Default_Second	MsgBox_Default_Second (256)	Second button is the default
MsgBoxCustom_Default_Third	MsgBox_Default_Third (512)	Third button is the default
MsgBoxCustom_Default_Fourth	#300	Fourth button is the default
MsgBoxCustom_Default_Fifth	#400	Fifth button is the default
MsgBoxCustom_Icon_Exclamation_Mark	MsgBox_Exclamation_Mark_Icon	Displays the exclamation icon
MsgBoxCustom_Icon_Information	MsgBox_Information_Icon	Displays the information icon
MsgBoxCustom_Icon_Question_Mark	MsgBox_Question_Mark_Icon	Displays the question mark icon
MsgBoxCustom_Icon_Stop	MsgBox_Stop_Icon	Displays the stop icon
MsgBoxCustom_Return_One	1	
MsgBoxCustom_Return_Two	2	
MsgBoxCustom_Return_Three	3	
MsgBoxCustom_Return_Four	4	
MsgBoxCustom_Return_Five	5	

Notes If you do not include one of the **MsgBoxCustom_Icon_** values, no icon is displayed.

The **MsgBoxCustom_Return_** global constants can be used to return and test the value of the clicked button.

If the application is not a GUI application, error 14078 (*A gui action has been requested in a non-gui environment*) occurs. If the number of button captions is zero (**0**) or greater than **5**, error 1407 (*Invalid argument passed to method*) occurs. If a caption is not a string or is null (**""**), error 1000 (*Invalid parameter type*) occurs.

Handling Message Boxes (PAR 67849)

The handling of the custom message box has been changed, as follows.

1. The application name is shown as the title if the value of the **Application** class **msgBoxCustom** method **title** parameter is null.

2. The icon required, specified in the **flags** parameter, is verified. If it is not valid, an invalid parameter exception is generated.

3. Very large strings without spaces are handled in the title and the message box text so that the message will not enlarge beyond two-thirds of the width of the monitor on which it is displayed.

Long titles are truncated. Long message text without spaces is wrapped onto the next line when strings exceed the width of the assigned message area.

The handling of ordinary message boxes has also been changed to validate the button and the icon specified in the **flags** parameter of the **Application** class **msgBox** method call. Calling the standard Microsoft message box API will silently fail if those flags were invalid. If not valid, an invalid parameter exception is generated.

Object Class creationTimeUTC Method (PAR 67561, JAD-I-536)

The **Object** class now provides the **creationTimeUTC** method, which returns the date and time at which the receiver was created as a Coordinated Universal Time (UTC) timestamp value.

Pasting into a JadeRichText Control (NFS 67765)

You can now programmatically paste from the Windows clipboard into a **JadeRichText** control.

- The **canPaste_** method returns whether there is content such as text or an image in the Windows clipboard that can be pasted into the **JadeRichText** control. (You can also obtain this status by calling the existing **JadeRichText** class **getRedoAndUndoState** method.)

- If there is suitable content such as text or an image in the Windows clipboard, the **paste_** method pastes that content into the **JadeRichText** control at the current cursor position. If the clipboard does not contain suitable content, the method does not result in any change. (This method is equivalent to selecting the **Paste** command in the context menu of the **JadeRichText** control at run time.)

Before you call the **paste_** method, call the **canPaste_** method to confirm there is suitable content available.

Package Initialization (PAR 66629)

The [JadeServer] and [JadeClient] sections of the JADE initialization file can now contain the **PackageInitializationDisabledApp<*n*>** parameter, which has the following string value.

```
user-schema-name,RootSchema-application-name
```

When the **PackageInitializationDisabledApp<*n*>** parameter is defined, package initialization is disabled when the specified RootSchema application is run from the specified user schema.

The ***<n>*** variable in the parameter name indicates a unique number; for example:

```
PackageInitializationDisabledApp1 = UnitTestSchema,JadeUnitTest
PackageInitializationDisabledApp2 = UnitTestSchema,JadeUnitTestRun
```

In this example, package initialization is disabled when either the **JadeUnitTest** application or the **JadeUnitTestRun** application is run in the **UnitTestSchema** schema.

Tip You can disable package initialization if the application does not require functionality from any packages imported by the user schema. This can reduce the startup time for the application.

Partial-Word Searches in HTML5 Online Help

The JADE HTML5 Web format online help now supports partial-word searches.

You can type part of a word or any string (including numbers) in the **Search** text box, press Enter or click the **Search** button, and then see the search results that match those characters.

Notes You must type at least six characters before you see partial-word search results.

As a partial-word search is not limited to words only, it works with any string (including numbers) that starts and ends with a space. For example, if you want to locate the TCP/IP address **143.57.055.259**, typing **143.57** into the search field and then pressing Enter finds the topics containing that string.

Print Preview Form and Skin Compatibility (PAR 67350)

When a form or control skin background color is defined with a brush rather than a color, the skin brush is always drawn, even if the value of the **backColor** property of the form or control is not the default. This conflicts with the behavior where the form or control **backColor** is used to draw the background and the skin **backColor** is ignored if the skin **backColor** is defined as a color and the form or control **backColor** value is not the default.

In earlier releases, when such a skin was used to preview the printing of a form and the skin had a white brush, the boundary of the printed output preview was not evident.

This issue has been addressed by making the behavior of the drawing of the skin background consistent, regardless of whether it is defined with a brush or a color. In both cases, if the form or control is defined with a non-default background color, that color is used to draw the background and the skin background definition is ignored.

Printer Methods Client Execution (PAR 66745)

The **clientExecution** method option is no longer added to the **Printer** class **getAllPrinters** and **getAllPrinterPaperSources** methods. (These methods, which are effectively type methods, pre-dated the availability of the **typeMethod** method option in JADE.)

When the **getAllPrinters** method is executed in a **serverExecution** method, it now returns the list of printers attached to the server node. When executed in an application server or standard client, the method returns the list of printers visible to the client device.

When the **getAllPrinterPaperSources** method is executed in a **serverExecution** method, it now returns the list of paper sources for the specified printer attached to the server node. When executed in an application server or standard client, the method returns the list of paper sources for the specified printer attached to the client device.

Regular Expression (Regex) Pattern Matching (JAD-I-438)

JADE now provides the **JadeRegexLibrary** class, which is the abstract superclass of the regular expression (Regex) pattern-matching Application Programming Interface (API) subclasses in JADE. This API reduces hand-crafted string parsing and code manipulation, to assist in the reading and testing of your code.

JADE does not implement a Regex engine itself, but wraps an existing implementation (the Per Compatible Regular Expressions (PCRE) library) with defined behavior and documentation. JADE Regex therefore uses the PCRE dialect, whose documentation that can be found at https://www.pcre.org/current/doc/html/pcre2syntax.html.

The JADE regular expression pattern-matching provides:

- The ability to find words and numbers, and their positions in a body of text based on a pattern
- The ability to replace a word or number in a body of text based on a pattern with a substitute word or number
- Extraction of data, using a pattern to extract data fields from a string
- Validation of data; for example, checking that a credit card number is in the correct format

- Command line **Key=Value** pair parsing
- Log error message parsing, particularly in a test framework

The subclasses of the **JadeRegexLibrary** superclass are summarized in the following table. (For details, see Volume 1 of the *JADE Encyclopaedia of Classes*.)

Class	Description
JadeRegex	Contains type methods for quick, simple use of the **JadeRegexLibrary**. Each method has common options that suit many use cases.
JadeRegexCapture	A capture group of your regular expression, containing information about it; for example, the text it matched, the group name, length, and so on
JadeRegexMatch	A single match of your regular expression against the subject string. It optionally contains **JadeRegexCapture** objects if capturing groups is enabled.
JadeRegexPattern	A compiled Regex object that provides enhanced functionality and performance over the **JadeRegex** class and its type methods.
JadeRegexResult	An object representing one or more matches resulting from a Regex operation.

In addition, the **JadeRegexException** subclass of the **NormalException** class is the transient class that defines details for exceptions that occur as a result of JADE regular expression pattern matching.

REST Client (NFS 65594, JAD-I-97)

In earlier releases, JADE provided RESTful web services using the **JadeRestService** class, but they could not be consumed other than using the limited functionality of the **JadeHTTPConnection** class **sendRequest** method.

JADE now provides JADE REST client classes, providing easy consumption of REST services The interface is similar to REST clients in other technologies; for example, the .NET RestSharp library (http://restsharp.org/).

This feature enables you to set properties on a **JadeRestRequest** object to specify the required resource and any required inputs such as parameters or a serialized object in the HTTP body (used for **PUT** or **POST** operations), then pass the object as a parameter to a method of the **JadeRestClient** class, which will return a **JadeRestResponse** object with properties for the various response information, along with deserialization methods.

In this release, the REST client proxy classes support the **GET**, **PUT**, **POST**, **DELETE**, and **OPTIONS** verbs, although other verbs are available when using the **JadeRestClient** class directly.

The new REST client and proxy classes are summarized in the following table. (For details, see Volume 1 of the *JADE Encyclopaedia of Classes*.)

Class	Description
JadeRestClient	Represents the client that sends the REST request to the server
JadeRestProxy	Grouping class for auto-generated proxy classes that model a REST API based on an OpenAPI specification
JadeRestDataModelProxy	**JadeRestProxy** subclass grouping auto-generated proxy classes that model the data structure of an imported REST API
JadeRestResourceProxy	**JadeRestProxy** subclass grouping proxy classes that expose the resource methods of the imported REST API
JadeRestProxyHook	Provides hook methods that can be reimplemented in your client, request, and response REST subclasses
JadeRestRequest	Represents a REST request that is to be sent to a REST API specification
JadeRestResponse	Contains the results of a request to a REST API endpoint

JADE also provides a proxy class generator, which consumes an OpenAPI specification of a REST API and generates the appropriate data model and resource proxy classes. Use the JADE development environment External Components Libraries Browser to import and maintain OpenAPI specifications. For details, see "Maintaining OpenAPI Objects", in Chapter 16 of the *JADE Development Environment User's Guide*. See also "JADE REST Client" in the *REST Services White Paper*.

REST Services

This section describes the REST service changes in this release.

REST Service Exceptions (PAR 66669)

Prior to JADE version 2018 (1800), the format of REST service call exceptions was XML. In JADE version 2018 (1800) and later, the format of REST service call exceptions is formatted in the style specified in the URL.

The default format of REST service call exceptions is XML. You can control the format used to generate REST service call exceptions by using the new **JadeRestService** class **exceptionFormat** property, which has an Integer value.

You can specify the required **JadeRestService** class **OutputFormat_** constant value in the **exceptionFormat** property (for example, in the **create** method of your **JadeRestService** subclass) if you want to change the exception output format at run time. The class constant values are listed in the following table.

JadeRestService Class Constant	Value	Exceptions are in...
OutputFormat_Json	0	JSON (Microsoft JSON) format
OutputFormat_Xml	1	XML format (the default)
OutputFormat_Json_NewtonSoft	2	JSONN (NewtonSoft JSON) format

Any other value (for example, **-1**) means that exceptions are returned in the format controlled by the received URL.

Applies to Version: 2018.0.02 (Service Pack 1) and higher

REST Service Security (JAD-I-430)

In earlier releases, RESTful web services APIs were not able to be secured; that is, anyone with the URL could consume it without providing authentication.

JADE now provides the ability to restrict JADE REST APIs so that only clients with a valid bearer token can consume the API. In addition, an API developer can customize the rules on what constitute a valid token to meet your requirements.

REST service security allows for the validation of token signatures, including asymmetrical tokens (for example, RS256) signed from third-party Auth providers. It also allows for the generation of symmetrical tokens (for example, HS256) and the association of required claims such as access level or token expiry against specific REST service methods.

The supported token is JSON Web Token (JWT) – an open standard tracked by RFC 7519. The JWT standard defines a compact and self-contained way for securely transmitting information between parties represented as a JSON object.

The new Add JSON Web Token Claims dialog enables you to specify the claims that must be present in a JSON Web Token in order to access a JADE REST API method. As long as one or more required claims are associated with a method, any incoming REST request must include a JSON Web Token in the authorization header of the HTTP request; that is, it must include a header of the form **Authorization: Bearer <Token>**. For details, see "Associating Required JSON Web Token Claims with REST API Methods" under "REST Service Security", in Chapter 2 of the *JADE Object Manager Guide*.

The new REST security classes are summarized in the following table. (For details, see Volume 1 of the *JADE Encyclopaedia of Classes*.)

Class	Description
JadeRequiredClaimAnnotation	Abstract class that represents an annotation on a **JadeRestService** REST API method, requiring a claim to be present in a JSON Web Token (JWT) and the claim to fulfill the **validateToken** method so that a client can access the associated REST method
JadeRequiredDelegateClaimAnnotation	Represents an annotation on a **JadeRestService** REST API method, validating the token is done by calling the method referenced by the **delegateMethod** property
JadeRequiredOneOfValueClaimAnnotation	Represents an annotation on a **JadeRestService** REST API method, validating the token by comparing the claim in the JWT to each of the values in the **allowedValues** property
JadeRequiredSingleValueClaimAnnotation	Represents an annotation on a **JadeRestService** REST API method, validating the token by comparing the claim in the JWT to the value contained in the **expectedValue** property
JadeJWTModel	Abstract grouping class for JSON Web Token (JWT) classes
JadeJWKSAuthProviderResponse	Can be used as the first parameter to the **parse** method of the **JadeJson** class
JadeJWTClaim	Represents one claim in a JSON Web Token
JadeJWTParser	Contains type methods used for parsing JSON Web Tokens
JadeJWTValidator	Contains type methods used for validating the signature claims of JSON Web tokens
JadeJsonWebKeySetReader	Provides methods to obtaining the public key from a JSON Web Key Set that is used to validate asymmetrically-signed JSON Web Tokens (JWTs)
JadeJsonWebToken	Represents a symmetrically-signed JSON Web Token that can be used by a JADE REST service to generate authorization tokens for its clients

To increase REST service security, use one of the following **jadeDevelopmentFunctionSelected** function security hooks.

Task Name	Entity Name	Description
applyRestSecurity	*Schema-name::type-name::method-name*	Applies security to a REST Service method
importOpenAPI	*Schema-name*	Imports (adds) an OpenAPI specification
removeOpenAPI	*Schema-name*	Removes an OpenAPI specification

In addition, the:

- **JadeRestService** class now provides the following methods
    - **addBearerToken**, which adds a bearer token (for example, a JSON Web Token) to the REST request
    - **fetchJWT**, which returns the bearer token from the Authorization: Bearer HTTP header of the incoming REST request
    - **fetchSecret**, which returns the secret with which to validate symmetrically-signed tokens
    - **getTargetMethod**, which gets the name of the method targeted by the incoming REST request
    - **validateShadowMethod**, which returns **true** if the method is a valid shadow method of a REST service method
    - **validateToken**, which validates a JSON Web Token against the required claims associated with the specified method
- **JadeRestService** class now provides the following class constants
    - EncryptionAlg_HS256
    - EncryptionAlg_HS384
    - EncryptionAlg_HS512
    - EncryptionAlg_RS256
    - ServerVariable_AllHttp
    - ServerVariable_AllRaw
    - ShadowMethodPrefix
- **JadeHTTPConnection** class now provides the following class constants
    - AuthType_Basic
    - AuthType_Bearer
    - HttpResponse_Created
    - HttpResponse_Forbidden
    - HttpResponse_NotFound
    - HttpResponse_Success
    - HttpResponse_Unauthorized
- **TimeStamp** primitive type now provides the following constant and methods.
    - **UnixEpoch** constant
    - **getSecondsFromUnixEpoch** method, which returns the number of seconds between the Unix epoch and the TimeStamp
    - **setFromUnixEpoch** method, which sets the TimeStamp by adding the specified number of seconds to the Unix epoch

RPS Diagnostic Dump on SQL Update Error

The [JadeRps] section of the JADE initialization file now contains the **DumpOnSqlErrorDuringCud** parameter, which when set to **true**, takes a diagnostic dump if an SQL create, update, or delete operation fails. (The default value is **false**.)

Security

For information about the security changes in this release, see the following topics elsewhere in this document.

- Security Restrictions
- Inspector Security
- Thin Client Security (PAR 66521)
- REST Service Security (JAD-I-430)
- SSL End-to-end, including SDS, Encryption (JAD-I-580, JAD-I-275)

Skins

This section summarizes the skins and **JadeSkinControl** subclass changes in this release.

Changing the Skin of a Control Scroll Bar (NFS 67522)

Scroll bars were previously drawn using the scroll bar skins attached to the application skin that was applied. By default, scroll bars are drawn using the **JadeSkinHScroll** and **JadeSkinVScroll** skins assigned to the application. You can apply a different skin to a form by calling the **Form** class **setApplicationSkin** method, which allows different scroll bars skins to be assigned for the form and its control children.

You can now assign a specific horizontal and vertical scroll bar skin to JADE controls that display scroll bars; that is, to the **BaseControl**, **ComboBox** list box, **ListBox**, **Picture**, **JadeRichText**, **Table**, and **TextBox** control classes and to the **Form** class.

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

- **myHorizontalScrollBarSkin**, which defaults to null
- **myVerticalScrollBarSkin**, which defaults to null

These properties, which you can set on the Jade Skin Maintenance dialog, apply only to the controls specified above (that is, the **BaseControl**, **ComboBox** list box, **ListBox**, **Picture**, **JadeRichText**, **Table**, and **TextBox** controls and to the **Form** class), and are ignored for any other controls.

The new Scroll Bar Skins group box on the Jade Skin Maintenance dialog **Form** and **Control** sheets contains the **Horizontal** and **Vertical** combo boxes, which allow specific horizontal and vertical scroll bar skins to be assigned to the **Form** skin and to specific **Control** class skins.

The rules for the use of the scroll bar skins are as follows.

1. For a form, if the form:

   - Has a specific scrollbar skin assigned of the required type, the form scroll bar is drawn with that skin.
   - Does not have a specific scroll bar skin assigned, the application scroll bar skin of the required type is used. If the application does not have a scroll bar skin assigned of the required type, the scroll bar is not skinned.

   When the form is an MDI frame, the MDI client window that hosts the child forms uses the form's scroll bar skin using the same rules.

2. For a control, if the control:

   - Has a specific scrollbar skin assigned of the required type, the control scroll bar is drawn with that skin.
   - Does not have a specific scroll bar skin assigned, the form scroll bar skin rules in step 1 of this list apply.

3. For a **ComboBox**, if the combo box:

   - Has a list box skin assigned, the above scroll bar rules for a control in step 2 of this list are applied to the list box part of the combo box display.

   - The list box of a combo box is skinned only if the **ComboBox** skin has a list box skin assigned to the **myListBoxSkin** property.

Cloning Skins (NFS 67523)

The Jade Skin Maintenance form now allows you to select an existing skin item and then create a clone of that skin using another name.

The **Applications**, **Controls**, **Forms**, **Menus**, **Simple Buttons**, and **Window State Images** sheets now all have a **Clone** button, which enables you to create a new skin item as a copy of an existing skin item. This functionality works as follows.

1. Select an item on the current skin list; for example, an application skin in the **Application Skins** list box on the **Applications** sheet.

   a. Click the **Clone** button. This clears the skin **Name** text box and displays a hint in red; for example, <Enter name of new application>. All of the displayed controls of the previously selected skin remain displayed in the same state (except for the **Name** text box).

2. Focus is then set to the skin **Name** text box, so that you can specify the name of the new skin.

3. Click the **Update** button. A new skin is then created as a duplicate of the existing skin, except that it has another name.

Note If the selected skin had been changed but not saved when you click the **Clone** button, you are prompted to save the existing changes for the selected skin.

If you click the **No** button, the changes are not be saved against the original skin but the changes remain for the new skin. If you click the **Yes** button, the changes are saved against the original skin *and* remain for the cloned new skin.

Foreground Color of Table Columns and Rows (NFS 67292, JAD-I-443)

The **JadeSkinTable** class now provides the **fixedColumnsForeColor** and **fixedRowsForeColor** properties, which enable you to set the color with which text of fixed text color of cells in a fixed cell. These properties are ignored if the default value is **#80000000** or a fixed cell has a specific foreground color set by the **foreColor** property of the **Table** class for a cell, row, or column.

The **fixedRowColorHasPrecedence** property of the **JadeSkinTable** class has changed, so that it now specifies whether cells that are in both a fixed row and a fixed column are drawn using the value of the **fixedRowsBackColor** and the **fixedRowsForeColor** properties. The default value is **true**; that is, cells that are in both a fixed row and a fixed column are drawn using the value of the **fixedRowsBackColor** and the **fixedRowsForeColor** properties. When the value is **false**, cells that are in both a fixed row and a fixed column are drawn using the value of the **fixedColumnsBackColor** and **fixedColumnsForeColor** properties.

If a fixed cell has a specific:

- Background color set by the **backColor** property of the **Table** class for a cell, row, or column, the skin background color value specified by the **fixedRowColorHasPrecedence** property is ignored when drawing the background of a fixed cell.

- Foreground color set by the **foreColor** property of the **Table** class for a cell, row, column, the skin foreground color value specified by the **fixedRowColorHasPrecedence** property is ignored when drawing the text of a fixed cell.

The **Table** control types on the **Controls** sheet of the Jade Skin Maintenance dialog now also allows these color values to be specified in the same way as other color properties.

Skin Category Selection in the JADE Painter (NFS 67295)

When you click on the **skinCategoryName** property on **Common** page of the Properties dialog in the JADE Painter, a combo box with a list of skin category names is now displayed.

The list box contains only those category names that are assigned to a skin of the same type or superclass type as the window currently selected in the Painter. The list box also contains a blank entry, to enable you to clear the current skin category name.

When the required skin has not been loaded into the current development environment, you can also specify a skin category name that is not displayed in the list.

Skinning Folder Overflow Buttons (NFS 67506, JAD-I-280)

When the **Folder** class **tabsLines** property was set to **TabsLines_SingleLine** and not all sheet tabs could be displayed in earlier releases, JADE added two arrow buttons on the right of the tab line. Clicking the buttons scrolled the sheet tabs left or right.

The display of these buttons has been changed as follows.

- The left arrow button is displayed on the left of the tabs line.
- The sheet tab that is farthest left is positioned to the right of the arrow.
- The right arrow is displayed at the end of the tab line, possibly obscuring part of the last sheet tab that is displayed.
- The arrows are drawn bordered with a black box and the background color is the value of the **backColor** property of the folder. Each box is the height of the tab line.

In addition, these buttons can be skinned. The **JadeSkinFolder** class has the following new properties.

- **myTabScrollLeftButton**, which defaults to null and is of type **JadeSkinSimpleButton**
- **myTabScrollRightButton**, which defaults to null and is of type **JadeSkinSimpleButton**
- **tabScrollButtonBackColor**, which defaults to **#80000000** and is of type **Integer**

The Jade Skin Maintenance dialog enables you to set the:

- Tab scroll left and right buttons to a **JadeSkinSimpleButton** set of images that define the normal, rollover, disabled, and down states.
- The background color of the button area.

If a folder skin does not specify a button image, the appropriate default arrow is displayed.

When the button image is set, the width of the normal state image defines the width of the button area.

If a button image is *not* defined, the value of the **tabScrollButtonBackColor** property is ignored.

If the value of the **tabScrollButtonBackColor** property of a folder skin is:

- Not the default (that is, **#80000000**), the background of the button area is drawn using that color
- The default value (**#80000000**), the background of the tab area is drawn using the background color of the parent of the folder.

The button image is vertically centered in the tab line.

SSL End-to-End, including SDS, Encryption (JAD-I-580, JAD-I-275)

JADE now provides Secure Sockets Layer (SSL) encryption between:

- **jadehttp.dll** (IIS) and the application server (REST and SOAP web service providers)
- SDS primary and secondary nodes

For details, see "End-to-End SSL Encryption", in Chapter 2 of the *JADE Installation and Configuration Guide*.

Stretching an Image (PAR 67370)

Images that are mostly white or black lost their clarity when displayed in a **Picture** control using the **stretch** property and the picture control is smaller than the image. When an image was stretched, JADE set the Microsoft stretch mode to **COLORONCOLOR**, which meant all pixels were treated equally, and Windows dropped pixels when the image was reduced in size. The exception was for a 1-bit image, where JADE calculated whether there were more white or black pixels and then set the Microsoft stretch mode to **BLACKONWHITE** (white pixels are discarded instead of black pixels) or **WHITEONBLACK** (black pixels are discarded instead of white pixels).

JADE has now been changed so that if the image consists of more than 50 percent of white pixels, JADE now sets the stretch mode to **BLACKONWHITE**. Similarly, if the image consists of more than 50 percent of black pixels, JADE now sets the stretch mode to **WHITEONBLACK**. This change improves the display of small images; for example, plan drawings.

Type Class Methods

The **Type** class now provides the methods summarized in the following table, which are similar to those defined in the **Object** class. (For details, see Volume 2 of the *JADE Encyclopaedia of Classes*.)

These methods allow you to dynamically call type methods without specifically requiring an instance of the type, which was previously the case with the base implementation on the **Object** class.

Method	Sends the specified...
invokeIOTypeMethod	Target type method containing a variable list of parameters to the receiver type instance, after switching to the specified target context execution context
invokeTypeMethod	Target type method containing a variable list of parameters to the receiver type instance, after switching to the specified target context execution context
sendTypeMsg	Message (a valid type method) to the receiver type instance
sendTypeMsgWithIOParams	Message (a valid type method) to the receiver type instance
sendTypeMsgWithParams	Message (a valid type method) to the receiver type instance