Contents

Overview  ......................................................... 4

Introduction .................................................. 5

General Concepts ......................................... 6

Versioning ..................................................................................................... 6
Schema Contexts ............................................................................................ 6
Instantiation Phases ........................................................................................ 7
Transition to Latest Version .............................................................................. 7

Schema Evolution ............................................. 8

How Changes are Made.................................................................................... 8
History of Changes Can Make a Difference...................................................... 8
Versioning System Descriptions ................................................................. 9
Current and Latest Versions ......................................................................... 10
Structural versus Non-Structural Versioning ................................................. 10
Ripple Effects ............................................................................................... 10
Avoiding Non-Structural Versioning .............................................................. 11
Load Styles .................................................................................................... 11
Unversioning Schemas..................................................................................... 14
When is a Class Versioned? .......................................................................... 14
Changes to a Class that can Cause Versioning.............................................. 15

Development Environment ................................................................. 15

Creating a Schema Context ........................................................................ 15
Selecting a Schema Context ......................................................................... 16
Distinguishing Schema Context .................................................................. 17
Versioning Indicators .................................................................................. 17
Displaying Versioned Entities ..................................................................... 20
Making Changes to the Current Schema Context ....................................... 21
Making Changes to the Latest Schema Context ......................................... 21
Unversioning a Schema ............................................................................... 21
Patch Control ............................................................................................... 22
Deltas ........................................................................................................... 22

Schema Instantiation .................................................. 22

Overview ...................................................................................................... 22
Database Reorganization .............................................................................. 22
Object Mutation ........................................................................................... 23
Inter-Object Conversion ............................................................................... 23
Application Deployment Cycle ..................................................................... 24
Online Phases .............................................................................................. 24
High Availability Deployment ..................................................................... 25

Copyright 2006, Jade Software Corporation Ltd.  All rights reserved
# Schema Instantiation, (continued)

- Database File Reorganization ................................................................. 25
- Database File Copy .................................................................................. 26
- Online File Reorganization ..................................................................... 26
- Online Reorganization Phases ................................................................. 26
- Multiple Worker Threads ........................................................................ 27
- Database File Compaction ....................................................................... 28
- Offline File Reorganization ...................................................................... 28
- Offline Phases .......................................................................................... 28

## Recovery Considerations

- Recovery Options .................................................................................... 30
- Restarting a Schema Instantiation .............................................................. 31
- Abandoning a Schema Instantiation .......................................................... 31
- Enable Archival Recovery ........................................................................ 31
- Replayable Database Reorganization ......................................................... 32
- Changes Requiring an Early Transition ...................................................... 32

## SDS Considerations

- Secondary Database Replay .................................................................... 33
- Secondary Database Availability ............................................................... 34

## RPS Considerations

- Instantiation Strategy ............................................................................. 35
- Alter Scripts ............................................................................................. 36
- Data Pump Restart ................................................................................... 37
- Manual Table Redefinition ..................................................................... 37
- Historical Table Handling ....................................................................... 37
- Table Consistency Checking ................................................................... 38

## Deployment Scenarios

- Scenario 1 .............................................................................................. 38
- Scenario 2 .............................................................................................. 39
- Scenario 3 .............................................................................................. 40
- Scenario 4 .............................................................................................. 41

## Summary

- 42

## Appendix A       JADE 6.1 Differences

- Schema Evolution Changes .................................................................... 43
- Development Environment Changes ......................................................... 44
- Schema Instantiation Changes ................................................................. 44

## Appendix B       How to Work With These Features

1. Choose the Correct Schema Context (Development) .............................. 45
2. Schedule Schema Instantiations (Development) ...................................... 45
3. Use the Appropriate Load Style (Deployment) ....................................... 45

## Glossary

- 46
Schema Evolution and Instantiation

Overview

Schema evolution is the process of changing the specifications or description of a system. Schema instantiation is the process of bringing those changes to life and making the set of changes current.

Increasingly, systems run 24/7 and changes to the system must be achieved in narrow time windows. These features have been designed to increase the availability of the system, both in development and in deployment. In development, objects can still be accessed even if their specifications are being changed. In addition, a developer can make changes to the system specifications and then deploy them into a running system with as little downtime as possible.

This paper discusses how developers take advantage of these features during development and when deploying system changes. Appendix A outlines the significant changes made to these features in JADE 6.1. Appendix B offers guidelines on taking full advantage of the benefits of schema evolution and instantiation. The glossary provides brief explanations of technical terms.
Introduction

When you add a property to a class that has instances thereby changing its structure, the class and related structures are versioned. At this stage, there are two versions of the schema: the current version and the latest version.

Schema evolution enables:

- Developers to work in multiple schema versions at the same time
- Structural changes while an application is still running

The current version contains the version of the class before the property was added, and matches the current structure of objects in the database.

The latest version contains the version of the class after the property was added, although this 'pending' change has not yet been brought to life. The process in which the specification of a system changes is called schema evolution.

Schema instantiation results in:

- Improved availability of production systems (system available longer when changes deployed)
- Faster database reorganization

To bring to life the changes in the latest version, they must be made part of the current version. This process is called schema instantiation. It includes reorganizing the database. Importantly, much of this process can be carried out online. You need only to shut the system down for the final transition step.
General Concepts

This section introduces a number of the terms and general concepts, which are then discussed in detail.

Versioning

JADE uses objects to store the description of a system. These description objects are referred to as the metaschema. Entities such as schemas, classes, and properties are instances of classes in much the same way as for user data objects. In order to deal with schema evolution, and describe both the current state and future state of the system, JADE uses the concept of versioning. Suppose, for example, in the Customer class the name property is increased in length from String[30] to String[40].

To represent both states of the system, JADE versions parts of the metaschema. In particular, at least the entities representing the Customer class and the name property are versioned. (As explained in the next section, other entities are also versioned.) The old version is referred to as the current version and the new version is referred to as the latest version.

If an application has created, deleted, or accessed an instance of a user class, the class is in use. The Browse | Classes In Use menu command can be used to find which classes are in use and which processes are using them. Even if an application is using the Customer class, the above change could be made with the application continuing to use the current version of the class; that is, the one with the shorter String[30] version of the name property.

Schema Contexts

Once the system is versioned, you see two versions of a schema where a change has been made. These are referred to as the current schema context and the latest schema context. You can work in the current schema context and make changes to the current version of the system, or you can work in the latest schema context and make changes to the latest version.

Changes made in the current schema context may change the behavior of any running application. This is true when you load a schema or when you work in the development environment. Furthermore, not all changes made in the current schema context may be visible in the latest schema context.
Instantiation Phases

When you decide to instantiate this change and move to the longer version of the name, a database reorganization is needed. As explained in more detail in a later section, a reorganization has a number of phases. The initial phases include an analysis of the changes between the current and latest versions and a modification to the file where the instances of the class are stored in the persistent database. In the above example, additional space is required for the longer name in each Customer object. While these phases of the reorganization occur, access to the current version of instances of the Customer class is possible and continues to use the shorter String[30] version of name. These phases are referred to as the online phases of the instantiation.

The final phases of the instantiation include any inter-object changes (for example, if the property name had been used as a key in a dictionary, the keys would need to be changed) and finally a transition to the new version. This transition makes the latest version the current (and only) version, deleting the prior versions of any metaschema entities. During these last two phases, the Customer class must not be in use. These are referred to as the offline phases. After the transition, the Customer class can again be used and instances of the class now have longer String[40] names.

Note The term reorganization refers to the two phases of the instantiation where the instances in the persistent database are changed. Part of this occurs during the online phase, involving intra-object changes (such as making space for new attributes), and part during the offline phase, involving inter-object changes (such as populating new collections based on existing references).

Sometimes the whole of the instantiation phase is loosely referred to as a reorganization, although this is really only part of the instantiation.

Transition to Latest Version

You can continue to use the system throughout the evolution (either in development or loading a schema in deployment) and the first phases of the instantiation of a new system description. This greatly increases the availability of the system.

In many systems, the time to load the schemas and perform the online phases of the instantiation is a significant proportion of the total time to upgrade the system. If applications can remain available during these phases, the downtime of the system can be greatly reduced.

The change to the latest version of the system usually involves significant changes to an application; for example, new or changed forms, extra menus, and different business rules and functionality. As explained later, the decision to make the transition entails an explicit action by the deployer and the system continues to run using the current version until that time.
Schema Evolution

Schema evolution is the process of changing the specifications or description of a system. Such changes are made to a system either as a series of changes in the development environment or as a collection of changes when schema files are loaded.

How Changes are Made

After you make a single change in the development environment, you usually deal with the consequences of that change before making the next change.

For example, suppose you change the type of a property and as a result, a number of methods are recompiled, some of which are in error. You might edit and recompile these methods before making another change. At various points in this evolution, you can decide to instantiate the change. For example, after changing the property and fixing the methods in error, you could instantiate the change and test the changes.

When a schema file is loaded using Schema | Load... menu command in the development environment or using the JADE Loader for a deployed system, a collection of changes are made together. In a typical scenario, you take a snapshot of the specification of a system to consolidate a series of changes made in the development environment by extracting schema files. Such files contain the complete description of the development system as it stands at the time when the schema is extracted.

History of Changes Can Make a Difference

Instantiating a number of changes individually in a development environment may not produce the same result as consolidating these changes into a single schema load and instantiating the changes.

For example, suppose you delete a property in the development environment and instantiate the change. This updates all user objects and discards the values of the property. Suppose you add the property back to the class with the same name and a different but compatible type. When this is instantiated, the new property is added back to the existing user objects with the default value for the new property type.

However, if you combine these two changes into a single schema load, which is then instantiated, the result is somewhat different. Rather than a property deletion followed by a property addition, the change now involves changing a property type and, during instantiation, the original property value is retained and converted to the new type.

A slightly more complex example is the renaming a property. If you do this in the development environment using the Properties | Change... menu command, the property again retains any value it may have in user objects. However if you make the same change by extracting and loading a schema, the compiler has no way of knowing the property was renamed and treats the change as the deletion of one property and the addition of another, which discards existing values.

Issues such as these require you to consider the way changes are deployed. For a property rename, use a JADE command file to achieve the desired result.
Versioning System Descriptions

Part of schema instantiation is the process of restructuring instances of user classes to match an updated system description. A change to an attribute may require increasing or decreasing the size of objects, and adjusting collections that use the attribute as a key. A change to a reference may also require inverses to be changed. To perform the instantiation, the process needs the specifications of both the old and new systems. As explained, JADE uses the concept of versioning to store these two definitions. The meta-structure that defines the system specifications uses instances of system classes such as Schema, Class, Property, and Method.

For example, suppose a property named address is added to the class Customer in the schema ModelSchema. An instance of the Property class is created to represent details of the new property address such as its name (address) and type (String[40]). The class Customer is represented by an instance of the Class class.

To represent the two definitions of the class, one with and one without the new property, the corresponding instance of Class is versioned. This results in a clone of the instance; that is, a new object that also represents the Customer class. The original instance is the current version of the class and the new clone is the latest version. The new property is then added to the latest version of the class to represent a class with an additional property. When this is done, the instance of the Schema class that represents the specification of the ModelSchema schema is also versioned.

Adding address property to Customer class

The current version of the schema contains the class without the new property while the latest version contains the class with the new property. An explicit linkage exists between the current and latest versions of any entity. When the change is instantiated, the reorganization process can access both versions of the meta-structure to decide what changes must be made to user instances of the Customer class to make space for the new address property.
Current and Latest Versions

The original version of the object is referred to as the current version as it represents the existing state of the user objects. Until the change is instantiated, instances of the Customer class do not have the additional address property. When user applications, JadeScript methods, or workspaces are run they use the current version of the class. This means that after a property has been changed (and therefore versioned) applications can continue to access, change, create and delete instances of the changed class. Changing the system description can be done at any time and the current system can continue to be accessed.

A consequence of allowing changes to classes to be made while applications are running and using the classes is that applications can continue to run while new schemas are loaded. Furthermore, as explained later, they can continue to run unchanged for much of the instantiation of the new specification.

Structural versus Non-Structural Versioning

Allowing the system to continue to run unchanged when a change is made to the system specification may require more than just versioning the property that is changed. The term structural versioning refers to those metaschema entities that must be versioned to make the reorganization phases aware which objects require updating. The term non-structural versioning refers to the other parts of the metaschema that are versioned purely to ensure that applications can run unchanged.

In earlier releases of JADE only structural versioning was performed. Non-structural versioning has extended the types of metadata entities that may be versioned, including entities such as methods, constants and forms.

An entity is structural versioned if the instantiation process needs to update instances of user classes. For example, if you change the type of a property from Integer to Decimal then at least the property, its class and the schema are versioned. If the property is used as a key in a MemberKeyDictionary class, the collection class is also versioned.

An entity is non-structural versioned if a user application would run differently if it were not. For example, if you changed the type of a property, methods that use the property are recompiled and may be marked in error as a result. (For example it may have used the property in a way that is valid for an Integer but not for a Decimal.)

Therefore, under non-structural versioning, the method is versioned, along with the class or type where the method is defined, and the latest version of the method recompiled. Even if this version is in error, applications continue to run as before, because when a method is invoked, the current version (which was not recompiled) is used.

Ripple Effects

If the changed property is used in methods in subschemas, those methods along with their classes and the subschema are versioned. If the method or property is exported in a package, the exporting package entity is versioned along with any schemas that import the package.

Similarly, if the schema load contains a new definition of a method, the method is versioned and the new definition replaces the latest version of the method. This change could also ripple out to other entities. For example, if the signature changes, all methods that invoke the changed method are also versioned before being recompiled.
A change to a method can also result in structural versioning. For example, if the method is a condition used as a constraint by a reference, the reference (along with its class and schema) is versioned, to allow the instantiation phase to maintain referential integrity.

Avoiding Non-Structural Versioning

Non-structural versioning, to allow the application to run unchanged, does not come without some overhead. Not only must additional entities be versioned, but they must also be deleted when the instantiation completes. The JADE compiler attempts to mitigate this overhead during a schema load by only versioning methods if they have changed.

The method source being loaded is compared with the current source, and if they are identical, the method is not versioned. Not only is the method not versioned, but it is also not even compiled. This results in substantial savings during a large schema load where many methods remain unchanged.

However to make this saving, the source of the method must be available. In earlier releases of JADE, when an encrypted schema was loaded, the source of the method was not saved and therefore could not be compared on any subsequent schema load. By default, the load of an encrypted schema now saves the encrypted source of each method in the database, to allow the next encrypted schema load to determine if a method has changed and therefore requires versioning and recompiling.

The methods are regarded as unchanged only if the source is identical. Even a formatting change in a method is regarded as a change, and results in versioning and recompilation. If an unencrypted schema file is loaded into a schema that contains encrypted source, the methods are not regarded as identical. In this case, all existing methods are versioned and compiled, and the old encrypted source discarded. Similarly, if an encrypted schema file is loaded into a schema that does not contain saved encrypted source, all existing methods are compiled and the old plain-text source is discarded.

Use the **DontSaveSource** option in JADE Loader, if you do not want to save the encrypted source in the database. Choosing this option means that subsequent encrypted schema loads must version and compile all methods, including existing methods that have not changed. You cannot use encrypted source to recompile a method; its only purpose is for source comparison during the next encrypted schema load.

Load Styles

Under some circumstances when a schema is loaded, you may know that non-structural versioning is not necessary. For example, there may be no problem in shutting down the applications for an extended period while the load and instantiation is carried out. This was the only option prior to JADE 6.1 and may still be the best and fastest option in certain circumstances.

When you load a schema file, you can specify a load style to eliminate all non-structural versioning. In the development environment, you achieve this by selecting the appropriate load style from the Load Options dialog. The **loadStyle** parameter in the batch JADE Loader achieves the same result.

The load style options are explained in the following subsections.
**loadStyle=latestSchemaVersion**

The default **loadStyle** is **latestSchemaVersion** and is the option most commonly used. A later section on deployment scenarios gives examples of when you would use the other two load styles.

- Structural versioning performed
- Non-structural versioning performed
- No versioning performed, if schema does not exist
- Schema is versioned (unless previously versioned)
- Load into latest schema context
- Applications continue to run unchanged but behave differently because of the changes loaded

The following diagram shows what happens when you load a schema file that changes the **name** property of a class from **String[30]** to **String[40]** and changes the source code of the **release** method. The change to **name** causes the **setName** method to be recompiled. The source code for the **setName** method is unchanged.

```plaintext
setName(val : String) updating;
begin
    name := val;
end;
```

The source code for the **release** method is changed as follows.

```plaintext
release() : String;
begin
    return "6.1";
end;
```

The **name** property and the **setName** method that refers to it are versioned because of the structural change. The **release** method is also versioned but not for any structural reason.
**loadStyle=currentSchemaVersion**

This option should be used only in rare circumstances when the schema is already versioned. For example, suppose you wanted to load hot fixes to methods (and have them take immediate effect) but you do not want to instantiate the schema at this stage.

This option is similar to using the **noStructuralChanges** option in schema loads in JADE versions prior to 6.1.

- Changes requiring structural versioning cause load to abort
- Non-structural versioning *not* performed
- Load into current schema context
- Applications continue to run unchanged

Use this option to load method changes or to change transient-only classes (such as forms, for example).

The following diagram shows what happens when you load a schema file that changes the **release** method but does not attempt to change the **name** property (attempting such a change would abort the schema load).

![Diagram showing schema versioning](image)

The current version of the **release** method is changed regardless of whether the schema is versioned.

**loadStyle=onlyStructuralVersioning**

This option results in versioning behavior that is similar to JADE versions prior to 6.1.

- Structural versioning performed
- Non-structural versioning *not* performed
- No versioning performed, if schema does not exist
- Schema versioned, if existing objects are structurally changed
- Load into latest schema context, if schema is versioned
- Applications shut down before load and restarted after instantiation
The following diagram shows what happens when you load a schema file that changes the `name` property of a class from `String[30]` to `String[40]` and changes the source code of the `release` method.

Unversioning Schemas

The JADE development environment and the JADE Loader both allow the removal of the latest version of schemas to discard pending changes before those changes have been instantiated. To ensure metadata consistency additional schemas may also be unversioned. If the instantiation has started then the schema cannot be unversioned unless the instantiation is aborted.

When is a Class Versioned?

Whether a change causes the class to be versioned depends on two factors: the kind of change being made and the schema context in which the change is made. At the end of this section, there is a (non-exhaustive) list of changes that cause versioning to occur. However, versioning also depends on the context in which a change is made and whether the class involved has persistent instances.

The rules for a schema load are presented in the following table.

<table>
<thead>
<tr>
<th>Load Style</th>
<th>Whether class is versioned</th>
</tr>
</thead>
<tbody>
<tr>
<td>latestSchemaVersion</td>
<td>All changes cause versioning</td>
</tr>
<tr>
<td>currentSchemaVersion</td>
<td>No changes cause versioning</td>
</tr>
<tr>
<td>onlyStructuralVersioning</td>
<td>Class with persistent instances is versioned</td>
</tr>
<tr>
<td></td>
<td>Class with no (or only transient) instances is not versioned</td>
</tr>
</tbody>
</table>
Similar rules apply when you make changes in the development environment.

<table>
<thead>
<tr>
<th>Where changes are made</th>
<th>Whether class is versioned</th>
</tr>
</thead>
<tbody>
<tr>
<td>Latest schema context</td>
<td>All changes cause versioning</td>
</tr>
<tr>
<td>Current schema context</td>
<td>Class with persistent instances is versioned</td>
</tr>
<tr>
<td></td>
<td>Class with no instances (or only transient) instances can only be changed if not in use. It is not versioned.</td>
</tr>
</tbody>
</table>

Finally, changing or deleting a property used in a Relational Population Service (RPS) mapping results in the RPS mapping being versioned along with schemas that contain the RPS mapping. This is the case even if the change to the property does not cause it to be versioned.

**Changes to a Class that can Cause Versioning**

Some of changes that cause versioning to occur are:

- Changing the volatility of the class
- Changing the membership of a collection class
- Changing the tuning parameters (block size or population) of a collection class
- Adding, changing, or deleting keys for dictionaries
- Changing the `delete when emptied` option for a collection class
- Changing the type (real or abstract) or the lifetime of a class if that class or its subclasses has an RPS mapping
- Adding a non-virtual property
- Changing the length or scale factor of a primitive non-virtual property
- Changing the type of a property
- Changing the inverse definitions of a property
- Changing constraints on a property
- Deleting a non-virtual property
- Changing RPS mappings as a result of adding, changing, or deleting properties

**Development Environment**

This section highlights features of the JADE development environment that support working with multiple versions of a schema.

**Creating a Schema Context**

When a schema is not versioned, there is only a single schema context: the current schema context. In that case, when you make a non-structural change, nothing is versioned. Methods, forms, and other non-structural entities can be changed and then tested by executing **JadeScript** methods, workspaces, or running applications.
When you make a change that causes structural versioning, such as adding a property to a class with persistent instances, a confirmation dialog is displayed. If you confirm the change, the schema is versioned and all developers signed on to the system are notified with a message box.

The result is that rather than a single schema context, there are now two: the current schema context and the latest schema context. By default, the developer who causes the structural versioning is moved to the latest schema context and subsequent changes made may cause non-structural versioning.

You can also manually version a schema using the **Schema | Version...** menu command. This also produces two schema contexts.

**Selecting a Schema Context**

When there is a current and a latest schema context, you can choose either in which to make changes. Although structural changes can be made only to the latest schema context, non-structural changes can be made in either.

The Schema Browser is a key window of the JADE development environment. It is the starting point for any interaction that the developer makes with the source code. All other windows in the development environment are initially opened against a schema context that is selected in the Schema Browser.

If a schema is not versioned, there is a single entry in the schema hierarchy for that schema. However, if the schema is versioned, there are two entries in the Schema Browser for the schema: one representing the current context and the other the latest schema context. You select the schema context you want to work with from the browser before you open additional windows.

To differentiate the two entries for a versioned schema, the latest schema context is highlighted with a green background color in the browser. You can define the color used in the Preferences dialog. The latest schema context always appears on the line immediately before the entry for the current schema context.

The following picture shows the Schema Browser displaying a versioned schema.

Here, **ErewhonInvestmentsModelSchema** is versioned and appears twice in the Schema Browser. The first entry is highlighted with a green background, to indicate the entry for the latest schema context. The entry below it, which is not highlighted, is for the current schema context.
Distinguishing Schema Context

In the preceding picture, the Schema Browser is displayed using a JADE skin that is part of a skin set that is designed to provide strong visual feedback about which schema context a window is currently opened against.

When this skin set is used, any window that is opened against the current schema context is shown with a blue title bar, and a set of chevrons that point towards the left. Any window opened against the latest context of a versioned schema is shown with a different color title bar, and a set of chevrons that point towards the right. This behavior is consistent throughout the JADE development environment and JADE Painter. The title bar skin changes to reflect the schema context of the currently selected window, which is useful if you work with child forms maximized.

As the Schema Browser is always open, its border color changes based on the context of the currently selected schema. The following picture shows the skin set for a window opened against the latest context of a versioned schema (or, as in this example, the latest schema context of the ErewhonInvestmentsModelSchema is selected in the Schema Browser and is therefore decorated with the latest version skin).

Versioning Indicators

The skin applied to an open window indicates which schema context the window was opened against, but does not indicate what is versioned within the schema. This lower level detail is indicated by a set of icon overlays and color options. Properties, constants and methods that are versioned have an arrow overlay on their list icon to indicate if the versioned item is the latest or current version. The following table shows the icon variations used in the development environment.

<table>
<thead>
<tr>
<th>None</th>
<th>Current</th>
<th>Latest</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>![Icon]</td>
<td>![Icon]</td>
<td>![Icon]</td>
<td>Property, method, or constant is public</td>
</tr>
<tr>
<td>![Icon]</td>
<td>![Icon]</td>
<td>![Icon]</td>
<td>Property or method is protected</td>
</tr>
<tr>
<td>![Icon]</td>
<td>![Icon]</td>
<td>![Icon]</td>
<td>Property is read-only</td>
</tr>
<tr>
<td>![Icon]</td>
<td>![Icon]</td>
<td>![Icon]</td>
<td>Property is a key</td>
</tr>
<tr>
<td>![Icon]</td>
<td>![Icon]</td>
<td>![Icon]</td>
<td>Method contains one or more errors</td>
</tr>
<tr>
<td>![Icon]</td>
<td>![Icon]</td>
<td>![Icon]</td>
<td>Entity in the Methods List is a condition</td>
</tr>
</tbody>
</table>
If an item is versioned, it has an arrow superimposed on the icon.

- Gray and pointing to the left indicates the current version
- Green and pointing to the right indicates the latest version

To show the current and latest version of attributes, constants, and methods at the same time, select the View | Show Composite View menu command. In that view

- Any versioned class is highlighted with the user-defined versioned background color, which is light green by default. The same background color is applied to the latest version of any method, property, or class constant.
- Both versions of any versioned method, property, or class constant are displayed.
- In the current version context, methods, properties, or constants that have been added to the latest version context are displayed.

To provide additional feedback in the current version context, foreground colors, which can be specified in the Preferences dialog, indicate one of three versioning states:

- **Changed color** (which defaults to orange) indicates that a property or class constant has changed between the current and latest version
- **Added color** (which defaults to pink) indicates a new method, property, or constant added to current context.
- **Removed color** (which defaults to gray) indicates a method, property, or class constant that exists in the current schema context and that has been removed from the latest schema context. This element is removed from the current schema when the class is next reorganized.

The diagram on the following page shows the current and latest version of attributes, along with the new icon overlays. In the window, the **AddressableEntity** class has been versioned. The length of the **String** attribute **webSite** has been increased from 60 to 80 and a new attribute **webSiteRSS** has been added. As **Show Composite View** is selected, both the current and latest version of the properties and methods are shown in the Class Browser window.

If **Show Composite View** is not selected, only details of the current versions of the properties and methods are shown as the Class Browser was opened from the current schema context.
As well as the icon overlays, properties and methods are shown with a foreground color, which can be selected on the Preferences dialog, to indicate if the item has been added, changed, or deleted from the schema context against which the browser is opened.

In the previous window, the `zUpdateAddressableEntityCommon` method is shown as versioned (it was versioned because it references the modified `webSite` attribute). You can work and update either version of the method, but are warned that any change made to the current version is not automatically propagated forward to the latest schema context. If a method is not versioned, changes made to the current version are automatically available in the latest schema context. If you change a method in the latest schema context, the method is versioned.

The ability to work in either the current or latest version of a method enables you to maintain methods in the current schema context. Changes made to methods in the current context are available for use in the current runtime system, while changes in the latest schema context reflect structural changes that will be available after instantiation.
Displaying Versioned Entities

Windows are available that provide further information on versioned entities.

The **Browse | Versioned Methods** menu command opens a window that enables you to browse versioned methods. This is similar to the functionality of the **Browse | Changed Methods** menu command.

The **Methods | Compare Versions** menu command enables you to compare the current and latest version of a versioned method side-by-side.

The **Browse | Display Version Info** menu command enables you to display a complete table of entities that are versioned. You can control the name and type of entities displayed; for example, select **Class** from the **Type** combo box to limit the display to versioned classes or **JadeMethod** to limit the display to versioned JADE methods.

The following display of versioned entities results from changing the length of the name property in the **AddressableEntity** class of the **ErewhonInvestmentsModelSchema**.

<table>
<thead>
<tr>
<th>Entity Name</th>
<th>Type</th>
<th>Date Created</th>
<th>Class</th>
<th>Dictionary classes that use the property as a key</th>
<th>Methods that reference the property, classes containing these methods, and schemas containing these classes</th>
<th>RPS mappings that use the property</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The single property change causes the following to be versioned.

- Property that is changed
- Class containing the property
- Schema containing the class
- Dictionary classes that use the property as a key
- Methods that reference the property, classes containing these methods, and schemas containing these classes
- RPS mappings that use the property
Making Changes to the Current Schema Context

The only entities that cannot be changed in the current schema context are classes that are already versioned and RPS mappings. All other entities can be changed in the current and the latest context.

Changes made to the current context are not propagated to the latest context if the entity being changed is already versioned.

Entities added in the current schema context are always propagated to the latest context.

Making Changes to the Latest Schema Context

When working in the latest schema context, changes made to an entity that also exists in the current schema context cause the entity to be versioned.

Unversioning a Schema

To unversion a schema, select the Schema | Unversion menu command from the current schema context. The following dialog is displayed.

The selected schema and any dependent versioned schemas that would also be unversioned are displayed in the list.

Click the Details... button to display the entities that will be unversioned.

Click the Unversion button to unversion the displayed schemas.

Note  A schema cannot be unversioned if the latest version of any of its subschemas was created after the schema was versioned.

If a schema in the list cannot be unversioned, no schema in the list is unversioned. In this case, your only option is to carry out the instantiation.
Patch Control

When a schema is versioned, an entry is made in patch control against the current version number. Similarly, when a schema is unversioned, an entry is made in patch control. Changes in the current schema context and changes in the latest schema context are stored separately. When a schema is instantiated, all patch control information in the current schema context is merged with the latest schema context so that a complete history of all changes is maintained. When a schema is unversioned, all patch control information in the latest schema context is deleted along with the latest schema version.

Deltas

When a change is made in the latest schema context to a method that is checked out, the method and the checked out method are versioned.

If you use deltas for source comparison purposes only, consider versioning the schema instead. Deltas should only be used when you need to test your code changes before committing them.

Schema Instantiation

Overview

In a development cycle, schema evolution and instantiation can occur iteratively in any size steps. When test and production versions of the application have previously been deployed, periodically these systems must be upgraded to conform to a new specification (evolved schema definition).

When an evolved schema definition is applied to a JADE system, the system is upgraded from its current definition (current schema version) to a new definition (latest schema version). This upgrade transformation process is referred to as schema instantiation.

Although this section focuses on the schema instantiation process, any deployment scenario necessarily incorporates (and repeats) aspects of schema evolution that occurred during development. This section also explains how system availability is maximized by performing schema instantiation steps online wherever possible.

Database Reorganization

Database reorganization is the term used to describe the processes that alter persistent objects to make them compatible with an evolved schema definition.

Note  JADE usage of the term reorganization may differ from that of other Database Management Systems.

Database reorganization encompasses two operations that can be applied to objects; object mutation and inter-object conversion.
Object Mutation

Object mutation (or intra-object conversion) is the term used to describe an operation that converts an object’s structure into a form that is compatible with the class definition in the latest schema version. Generally, when an object is mutated its size changes as space in the object is adjusted, to reflect new or removed embedded properties. The following changes to class definitions are applied to instances of the class by mutation.

- Adding and removing attributes
- Adding and removing embedded references
- Adding and removing exclusive collections
- Changing the type of an attribute (any combination of type-to-type conversion)
- Changing the length of $\text{String}$ or $\text{Binary}$ attributes (includes converting embedded $\text{String}$ or $\text{Binary}$ primitive types to string large objects (slobs) or binary large objects (blobs), respectively, and the reverse, and truncation)
- Changing the precision or scale factor of a decimal attribute
- Moving blob and slob attributes up or down the class hierarchy
- Removing all inverses from a relationship (that is, after the change, there are no inverses between the properties that were involved in the relationship)

Inter-Object Conversion

An inter-object conversion process involves more than one object at a time and incorporates actions necessary to satisfy relationships between objects, satisfy constraints, populate new collections, and repopulate dictionaries when keys or other dictionary attributes are changed. The following schema changes are applied to objects during the inter-object conversion phase.

- Adding an inverse
- Removing some, but not all, inverses from a relationship (that is, after the change, inverses still exist between the properties involved)
- Constraint maintenance
- Inverse definition changes; for example, changing the inverse-required option
- Changes to dictionary specifications; for example, key definitions or the no-duplicates constraint
- Changes to the physical attributes of a collection; for example, block size and expected population
- Validation of the member objects of a collection against its member type definition
- Validation of reference property type changes
- Moving exclusive collections up or down the class hierarchy
Application Deployment Cycle

Deploying an application upgrade entails instantiating an evolved schema. This instantiation process can be broken down into phases, which are explained in detail in later sections. These phases are grouped into two categories.

- Online phases, so-called because they can be performed online if required
- Offline phases that must be performed offline

Reorganization Not Required

When an evolved schema is instantiated, a database reorganization may not be required. This case occurs when there are no structural changes that require persistent instances to be converted to conform to a new specification.

The following diagram depicts this simplified application deployment, which consists simply of a schema load followed by a commit step.

Online Phases

The following phases of an instantiation occur first and can be performed online.

- Schema and method compilation
- Forms definition file load

*Note* Changes to versioned Control subclasses are deferred to the offline phase.

- Database file reorganization

When these phases are performed online, the system remains available to users, and applications can access and update objects defined by the current schema version without restriction.
High Availability Deployment

The following diagram depicts the online phases of an application deployment, showing the three sequential phases of database file reorganization. As mentioned previously, there may be no need to perform the database file reorganization if there are no structural changes that affect persistent instances.

Loading schemas and performing database file reorganization online minimizes the system downtime required to deploy an application upgrade. Reorganized files can remain in dual update mode until your scheduled downtime window is reached.

You can perform any or all of the above steps offline, if required. As explained previously, you can avoid the versioning overheads that are necessary only to support an online instantiation, by specifying the following load style when loading schema files offline with the JADE Loader.

```
loadStyle=onlyStructuralVersioning
```

Database File Reorganization

Database file reorganization is a file-level operation that incorporates object mutation. When a database file is reorganized all objects contained in the file are processed. File reorganization reads from an existing database file creating a new file; it is never performed in-place. Objects that do not require conversion are simply cloned (copied) from the source file to the target file.

The file reorganization process reads objects from a current database .dat file, either cloning or mutating each object, and stores them in a new .reo file usually at a different offset in the file. Since file reorganization generally changes the location of objects, the physical indexes that map object identifiers to file addresses must be re-established in the target .reo file.

Database file reorganization can take place offline or online.
Database File Copy

All the objects in a database file may not require mutation, but may require processing in the later inter-object phase. In that case, the database file is simply copied rather than being reorganized. A database file is either reorganized or copied.

Online File Reorganization

Online file reorganization takes place when you opt to leave the database server and user applications running and also opt to allow updating transactions to continue concurrently with the reorganization process.

While online file reorganization is taking place, the system remains available to users and applications can access and update objects defined by the current schema version without restriction. This form of reorganization is best suited to systems that require high availability where it is beneficial to allow the system to run while the database reorganization process takes place.

Online Reorganization Phases

When database files are reorganized online (that is, concurrent updating transactions are permitted), each file is processed through three distinct phases or modes.

- File reorganization pass
- File synchronization
- Dual update mode

File Reorganization Pass

The online file reorganization process reads objects serially (not using an index) from a current .dat file cloning or mutating the objects into the target .reo file.

Since concurrent updating transactions may be updating objects while they are being cloned or mutated, the file reorganization process fetches objects from the shared database cache if resident, but avoids making non-resident objects resident.

This sequential processing of database files has some impact on normal processing since it competes for physical I/O resources.

File Synchronization

When the file reorganization (or file copy) pass is complete, there may be updates made by concurrent transaction activity that are not reflected in the target .reo file. The file synchronization phase makes use of database journals to apply updates to the .reo file that were made to the .dat file by transactions after the initial population of the .reo file had begun. The effects of transactions that aborted during the reorganization pass are of course removed.

File synchronization has two resource impacts. The first impact is that a redo operation using journal data requires that journals from the start of a file reorganization process remain current until file synchronization has completed. Completion of file synchronization allows journals that are no longer required to be released and moved to archive status. The second impact is I/O contention on the journal volume since file
synchronization activity reads journals randomly; while in normal processing, journals are written sequentially.

To minimize the journaling resource overheads while synchronizing files, once a file has been synchronized it is moved into dual update mode.

**Dual Update Mode**

When file synchronization is achieved, the `.reo` and `.dat` files are associated in a dual update mode so that updates to the current `.dat` file are also applied to the `.reo` file in combination with required object mutation operations. To allow the transition from synchronization to dual update mode to occur, write access to the file is momentarily (but not noticeably) blocked.

Files can remain associated in dual update mode indefinitely or until the database server is shut down. This allows sites to schedule the downtime window required to perform the offline phase of a schema instantiation.

The main processing costs incurred while the files remain in dual update mode are:

- **CPU**: Each object create, update, or delete operation is performed twice with object mutation, if required.
- **I/O**: Each object create, update, or delete operation is applied to two database files.

The latter overhead can be mitigated to some extent by locating the reorganization work files (`.reo` files) on a different physical volume to the main database files. This is accomplished using the `ReorgWorkDirectory` parameter in the `[PersistentDb]` section of the JADE initialization file.

**Multiple Worker Threads**

The database file reorganization mechanism posts jobs, in descending file size order, in a work queue for processing by worker threads.

Worker processing is complete for a file when it is ready to be instantiated; that is, it is in dual update mode in an online reorganization or it is populated in an offline reorganization. By default, a single worker thread is assigned to reorganize the database.

When multiple files are to be reorganized, parallelism can be achieved by increasing the number of workers, using the `ReorgWorkerThreads` parameter in the `[PersistentDb]` section of the JADE initialization file. The current `ReorgWorkerThreads` setting is also honoured during roll-forward recovery and during secondary replay.

The use of multiple worker threads is designed to exploit parallelism in the server hardware such as multiple CPUs, I/O channels, and disk spindles in the storage subsystem. If your database is not hosted on a disk subsystem that can support parallel I/Os, configuring multiple worker threads may be counterproductive if it results in increased contention for resources. Only you can determine by experimentation the best setting for your environment, considering factors such as hardware capability and configuration, and acceptable impact on other processing.
Database File Compaction

File compaction is a special form of file reorganization where all objects in the file are cloned and none are mutated. The main reasons you might consider compacting files are to reclaim free space and to achieve denser loading of physical indexes. A secondary objective might be to eliminate file system fragmentation that has accrued as a result of incremental file extension over a period of time. Any file reorganization pre-allocates the target .reo file to the larger of the defined initial file size or size of the current database file, to help avoid fragmentation.

A database file can be compacted online while permitting concurrent transactions to update the file. Moreover, the compaction operation can be aborted without losing updates made by committed transactions.

File compaction operations cannot be performed at the same time as a schema instantiation.

Offline File Reorganization

Offline file reorganization takes place when you disallow updates and is best achieved by shutting down the database server and performing all deployment steps in single user exclusive mode.

An offline reorganization is simpler to manage, consumes fewer resources, and generally completes in a shorter timeframe than an online reorganization. An offline reorganization is best suited to situations where high system availability is not essential; that is, the downtime required to perform the operation is acceptable to your business.

To ensure the reorganization is tailored to offline operation, you must specify that updates are disallowed when initiating the reorganization. In an offline reorganization, write access by user applications to all files involved is not permitted for the duration of the reorganization.

An offline file reorganization reads objects in index order from a current .dat file cloning or mutating the objects into the target .reo file. The offline process uses algorithms that assume the contents of the file are not changing during the operation and can use private file caches, bypassing the shared database object cache.

Offline Phases

The following steps or phases of a schema instantiation must be performed offline.

1. Initiate transition
2. File instantiation
3. Inter-object conversion
4. Commit step

Once the offline phase of a schema instantiation has been entered, the database is not available to applications until it completes.
The following diagram depicts a timeline of the steps involved in the offline phase of a schema instantiation.

Initiate Transition

- Instantiate Files
- Establish or Validate InterObject Relationships
- Commit

**Schema Transition**

Schema transition is the process that makes the latest schema version current; the transition itself occurs during the offline phase.

**Initiating the Transition**

When the start of the offline phase must be scheduled because it necessarily entails an interruption to normal processing, an explicit user action is required. This action is referred to as initiating the transition and can be accomplished through the Classes Needing Reorg dialog in the JADE development environment or by running a non-GUI jadclient application to execute JadeReorgApp from the command line.

Initiating a transition enters the offline phase from which point onwards instances of versioned classes are no longer available to applications until the instantiation completes. The transition can proceed only when no versioned schemas and classes are in use.

Any applications, including server applications, that are accessing the versioned schema or classes, must be shut down to allow the transition to proceed. If the transition is initiated but cannot proceed because running applications are using versioned schemas or classes, the instantiation is interrupted at this point. To continue, you must manually restart the schema instantiation when the schemas and classes are no longer in use.

A convenient way to gracefully terminate server applications and ensure the system remains offline for the duration of the offline phases involves shutting down the database server and initiating the transition in a single-user exclusive mode. For advice on how to do this, see the section on “Deployment Scenarios”.

**File Instantiation Step**

In this second step, all reorganized database files are instantiated, which makes them current.

File instantiation requires a database quiet point, which is a point where no transactions are active. All transactions that are committed during the online phase of the reorganization have been applied to all .dat and .reo files involved in the reorganization.

File instantiation involves the following actions.

1. Each current .dat file is renamed to .bak
2. Each new .reo file is renamed to .dat making it current

Each backup (.bak) file is retained until the final commit step of the instantiation to allow the reorganization to be aborted or abandoned without losing committed transactions.

**Inter-Object Conversion Phase**

The inter-object phase performs conversions involving multiple objects as described earlier in this section. In this phase, referential integrity as defined by the rules of the latest schema definition is restored. This phase also performs validation of inverses and checks that collection members remain type-compatible with the defined collection membership, if this has been changed.

The inter-object conversion phase employs standard object manager operations, which are normally fully audited and recoverable, as explained in a later section “Replayable Database Reorganization”.

**Commit Step**

In this final step, the following actions are performed within a single recoverable database transaction.

1. The transition to the latest schema version is performed
2. Prior versions of schema metadata are deleted

Once this transaction has been made durable (that is, its recovery is guaranteed in the event of a failure), the .bak files are removed.

Following the successful commit of this transaction, the changes from the latest schema version, which is now instantiated, become visible to user applications.

**Recovery Considerations**

Schema instantiation can fail for various reasons.

- The server could crash at any stage
- The inter-object phase could encounter a 'No duplicates' violation
- A lock exception could be encountered in the commit phase

In all failure scenarios, committed transactions are not lost; that is, transaction durability is preserved. In other words, when you choose to perform part of an application deployment online and the deployment fails for any reason, user transactions that were successfully committed during the deployment can be recovered even if the effects of the failed deployment are removed.

In most cases, an interrupted schema instantiation can be restarted; however, this may entail redoing steps that have already been done.

**Recovery Options**

When something goes wrong during a schema instantiation, you can either:

- Restart the schema instantiation
- Abandon the schema instantiation
The first option applies to scenarios where the schema instantiation was interrupted (for example, the machine halted) and reprocessing can still be accomplished in the remaining system downtime window.

The second option may be necessary in failure scenarios where there is no degree of certainty the problem that caused the failure can be resolved and the instantiation completed within the remaining system downtime window.

### Restarting a Schema Instantiation

When a database server node is restarted following a crash that occurred during a schema instantiation, restart recovery recovers all committed transactions including all transactions applied to the metaschema by schema and forms definition file loads.

An interrupted database reorganization is not automatically restarted as part of database recovery. Restarting an interrupted reorganization requires an explicit action. To restart an interrupted reorganization, use the **Schema | Reorg | Restart Reorg...** menu command in a JADE development environment or run a non-GUI **jadclient** application to execute **JadeReorgApp** from the command line, including the following argument.

```
action=restartReorg
```

Restarting an online reorganization interrupted during any of the online phases performs all file reorganizations again. Restarting an offline database reorganization reorganizes only files that have not been previously completed.

### Abandoning a Schema Instantiation

One way to recover from a failed instantiation is to restore from a prior database backup and then roll forward through available journals. However, there is an alternative. You can also abandon a failed instantiation using the following two steps

1. Abort the database reorganization using **JadeReorgApp**
2. Unversion schemas using the JADE Loader

For simplicity, the first step can be performed unconditionally even when there is no reorganization to abort. Examples of the command lines required for each of these steps are provided later, in the section on “Deployment Scenarios”.

Abandoning an instantiation using the above steps is generally preferable and in most cases is quicker than restoring and recovering from a backup.

### Enable Archival Recovery

The “Environmental Considerations for Deploying JADE” white paper discusses types of failure that can impact database processing and explains the importance of enabling archival recovery for business-critical systems.

If it is not acceptable to your business processing to lose committed transactions in the event of a catastrophic failure, you must operate your database with archival recovery enabled; that is, the **EnableArchivalRecovery** parameter in the [PersistentDb] section of the JADE initialization file is set to **true**.

You should leave archival recovery disabled only if it is acceptable to your business processing to lose all transactions processed since the last database backup. Archival recovery must be enabled to support the ability to replay a database reorganization.
Replayable Database Reorganization

The **Replayable** option is only valid when archival recovery is enabled. When your database is configured to support archival recovery, you need to select whether the reorganization is replayable or not; it is replayable by default.

When enabled, the **Replayable** option causes additional control information to be audited in the journal to support the following operations.

- Restarting an interrupted reorganization
- Re-applying, using roll-forward recovery, the effects of an instantiation to a database backup taken prior to the instantiation
- Replaying the effects of an instantiation on native SDS and RPS secondary databases

None of the operations listed above is possible if **Replayable** is not selected. The **Replayable** option generates significant auditing overheads only during the inter-object phase of a reorganization. The inter-object conversion process uses normal object manager transactions; when **Replayable** is selected, these transactions are audited in the journal. You can perform an online file reorganization that is not replayable without jeopardising user transactions that committed during the online phases; these are always audited and recoverable, even when the inter-object conversion phase is not.

Assuming you run your business-critical system with archival recovery enabled, only consider deselecting the **Replayable** option if you need to significantly reduce the elapsed time taken to perform the inter-object conversion phase in order to further minimize production system downtime windows. The best way to estimate the time to perform this operation and the reduction in elapsed time to expect with auditing is to perform a trial run on a test system, using representative data. You will need to extrapolate the results if test data volumes differ from production.

A non-replayable reorganization inserts a discontinuity record in the journal that when processed in roll-forward recovery or SDS replay, causes recovery or replay to halt with an audit discontinuity exception.

Changes Requiring an Early Transition

In the high availability deployment scenario mentioned previously, the transition occurs as late as possible in the schema evolution process, as shown in the following diagram.
With certain schema loads, the transition must occur before the forms file can be loaded, as shown in the following diagram.

The early transition is required if the schema load contains the following changes involving a user-defined control subclass.

- Adding, removing, or changing a property on a user-defined control subclass
- Adding a new user-defined control subclass

You can avoid unnecessary reorganizations when loading more than one schema by loading all schema files before forms files, using a multiple schema load file as follows.

```
#MULTIPLE_SCHEMA_EXTRACT
SchemaA.scm
SchemaB.scm
SchemaC.scm
SchemaA.ddb
SchemaB.ddb
SchemaC.ddb
```

## SDS Considerations

### Secondary Database Replay

A secondary database automatically replays all phases of a schema instantiation including any replayable database reorganization performed on the primary. A non-replayable reorganization, as the name implies, is not replayed by a secondary database; database tracking halts when it encounters the audit discontinuity record described in the previous section.

When database tracking has halted at an audit discontinuity, it is still possible for query applications to access the database. However, eventually you will need to re-establish the secondary database from a post-instantiation backup of the primary to enable database tracking to continue beyond the discontinuity.

While a primary database is open and active with updating transactions in progress, journals continue to be generated and transferred to connected secondary servers. A secondary database enters a reorganization mode when it replays the first reorganization control record. The replay of a reorganization on a secondary processes through several states that can be interrogated using the SDS administrative interfaces.
The first reorganization replay state is **Seeking Approval**. While in the **Seeking Approval** state, database tracking scans ahead through journals looking for specific reorganization control records, without replaying any reorganization operations or database updates. Essentially, replay is paused until the disposition of the reorganization process on the primary has been determined. Tracking remains in the **Seeking Approval** state until an **initiate transition** or an **abort reorganization** audit record is processed. The reorganization proceeds only when an **initiate transition** audit record is processed; this record signifies the online phases completed on the primary and an **initiate transition** audit record was executed. If instead, an **abort reorganization** audit record is encountered then the secondary splices out the failed reorganization and continues with normal replay, applying any updates that occurred during the failed or aborted reorganization.

### Secondary Database Availability

Replay of a schema instantiation on a secondary database also has an online and an offline phase. When an online reorganization is replayed on an SDS secondary, the database remains available to inquiry applications for the duration of the online phase of the schema instantiation. The inquiry applications can continue to access objects defined by the current schema version.

When the **initiate transition** audit record is replayed, the database tracker performs the offline phases of the instantiation, accomplishing a transition to the latest version of the schema definition as was done on the primary database. Initiating a transition has the same prerequisites as it did on the primary; that is, no versioned schemas and classes can be in use. To achieve this prerequisite, JADE attempts to gracefully shut down all user applications, including server applications, before proceeding.

The instantiation does not proceed until all user applications have been shut down, so if a user application has stalled and is unable to terminate, the instantiation cannot proceed. To resolve a stalled instantiation, an administrative user must intervene to arrange for applications to be terminated and nodes to be shut down, if required. If necessary, you can shut down and restart the database server at this point.

Once the schema instantiation has been replayed and committed, database server applications and application server applications configured to start automatically are restarted. The following diagram shows the phases occurring on the secondary database.
By default, the **initiate transition** audit record is replayed as soon as it is encountered. If you need to schedule when this occurs so you can arrange for a graceful termination of user applications, you should consider initiating a **managed transition**. The main difference in a managed transition is that a special **stop tracking** audit record is inserted in the audit stream. This causes secondary database tracking to halt just prior to starting the offline phases so that the required downtime for inquiry applications can be managed. The following diagram shows the phases in a managed transition.

Deferring replay by disabling tracking allows a failed schema instantiation to be spliced out on both the primary and secondary without the need to re-establish the secondary from a backup.

**RPS Considerations**

**Instantiation Strategy**

When an RPS mapping is versioned, a special **mapping check** audit record is written to the journal on the primary database as part of the instantiation commit transaction. The **mapping check** audit record contains a list of schemas and RPS mappings that were versioned. This audit record is recognized and replayed by RPS database nodes and ignored by native SDS databases. When an RPS node replays a **mapping check** record and a versioned RPS mapping matches the configured mapping, the current and latest versions of the RPS map are compared. Any differences are used to generate a Data Definition Language (DDL) script that can be applied to the target Relational Database (RDB) to restore schema consistency.

Replaying a schema instantiation on an RPS node configured for **Full** or **Mapped Extent** replication also performs any required reorganization to persistent objects replicated in the database. A database configured for **Working Set only** replication skips reorganization steps, but replays other required aspects of the instantiation.

Once a successful schema instantiation has been replayed and committed, the **Datapump** application and database tracking is halted. This halt in tracking occurs even if no changes to the target relational schema are required, in order to satisfy standard requirements for transition to a new schema version. The **Datapump** application and database tracking are automatically restarted when certain conditions, described below, are met.
Alter Scripts

A generated alter script contains SQL DDL commands to modify existing tables, columns and stored procedures that bring into effect the changes that were made in the RPS mapping or mapped classes in the primary database.

The alter scripts generated by an RPS node for instantiation are created with a name in the following format.

alterTable_<RPS-mapping-name>_<mapping-OID>.sql

The mapping OID qualifier is the object identifier of the RPS mapping entity and is appended to the file name to ensure uniqueness. The alter scripts are written to the Relational Database Path that you configure using the RPS Manager application.

Two types of alter script can be generated. The first is termed an auto initiate script and makes use of SQL alter table commands to modify tables in the target. The second is a manual script and is generated to assist with manual table redefinition when automatic table redefinition is not applicable.

A manual alter script contains DDL commands to rename any affected tables and create new empty tables. Comments are included at the top of the script that describe what has changed and what actions the script will take. The following is an example of this descriptive text.

-- The following change(s) to the table 'Agent'
-- new column 'webSite'
--
-- could cause an incompatibility between your JADE and
-- SQL databases. Therefore this table will be renamed
-- and a new one created. You will also need to extract
-- table data from the JADE database and load it into
-- this new table. Table renamed rather than dropped,
-- it can be dropped manually when no longer required.

The following JADE schema changes are achieved using alter table DDL commands.

- JADE property added or removed from a class that appears in an instantiated class mapping
- JADE property removed from an instantiated class mapping
- Selected JADE attribute to SQL Column type changes

The following JADE schema changes result in a manual alter script using a rename of the old table, and a create of the new table to achieve the table redefinition.

- JADE property added to an instantiated class mapping
- Filter constraint added to or removed from class mapping
- Filter constraint used in a class mapping changed
- Most JADE attribute to SQL Column type changes
Data Pump Restart

Once a schema instantiation has been successfully replayed, the Datapump application and tracking is halted. The Datapump application restarts automatically if the following conditions are satisfied.

- RPS Datapump application is configured to start automatically with the following setting in the JADE initialization file
  
  ```
  [JadeRps]
  AutoStartDataPump=true
  ```
- RPS node is configured to sign-on and authenticate with the target RDB server without user intervention
- RDB schema requires modification and the alter script is designated an auto initiate script or no changes are required to the RDB schema

When the Datapump application is restarted, auto initiate scripts are executed automatically without user intervention. Once an auto initiate script has been applied, consistency checking is performed (see below) and, if no errors are detected, database tracking and replication resumes.

Manual Table Redefinition

Where table redefinition cannot be fully accomplished using alter table DDL commands, an administrative user must then intervene to apply the table redefinitions, repopulate, and restart the Datapump application. An expert database administrator (DBA) may be able to accomplish the required changes in-place, avoiding the need to completely reload the table. If this is not possible or practical, execute the manual alter script generated by RPS directly without alteration, using the RPS Manager application.

To reload a table, follow these steps.

1. If you accept the default table replacement strategy, execute the generated alter script to rename the original table and create a new empty table
2. Extract objects for affected selected class maps from the primary database or the secondary RPS database, using the RPS Manager application
3. Execute the generated bulk insert script to load the extracted data

The RPS Manager application provides a menu command to execute the generated bulk insert script or to perform an extract and load as a single action.

Historical Table Handling

Historical mode tables are not redefined when mappings are changed but the current table version is renamed (by adding a date and time suffix) so that the latest version can retain the undecorated name that you specified in the mapping. It is the administrative user’s responsibility to archive and drop historical tables from the database when they are no longer required.
Table Consistency Checking

A table consistency check is performed each time the RPS node connects to the relational target or following the application of one or more table redefinitions.

The table consistency check imports the definition of all tables associated with class mappings for the database and checks that column names and types match those defined in the JADE schema. This consistency checking detects cases where a table definition or redefinition has not been applied or you have altered the definition of a target table for some reason, introducing a mismatch.

When a mismatch is detected, details about the mismatch are logged and the connection attempt fails. An administrative user must correct the mismatch before retrying the RPS connection.

Deployment Scenarios

Scenario 1

Consider an application where high availability is not essential but deployment needs to be completed as quickly as possible with minimal administrative intervention.

In this scenario, you would organize the new schema (.scm) and forms (.ddb) files into a multiple schema load (.mul) file so that schema files are loaded before forms definition files. Provided user-defined control subclasses are not marked for reorganization during the load, the instantiation is initiated after the files are loaded. If a user-defined control subclass requires reorganization, the instantiation must be performed before loading forms definition files.

1. Shut down the system.

   If possible, perform a graceful system shut down by signing off users. Stop server applications by shutting down the database server.

2. Backup the database (optional).

   Since you can restore from you last database backup and roll-forward to a point-in-time before the deployment failed, a pre-deployment database backup is not essential.

3. Load the schema and forms definition files and perform any reorganization.

   Use the JADE Loader batch utility to initiate the file load as follows
   
   ```
   jadloadb path=c:\jade\system ini=c:\jade\jade.ini server=singleUser
   schemaFile=myDeploy.mul reorgAllowUpdates=false
   loadStyle=onlyStructuralVersioning
   ```

4. Restart the system.

Post Deployment Backup

Roll-forward recovery cannot replay the effects of a non-replayable reorganization. A post-deployment backup is therefore highly recommended whenever a non-replayable reorganization is performed. If time permits, take an offline backup before restarting the system. Otherwise, restart the system and take an online backup.
Error Handling
The simplest procedure is to abandon the instantiation in the deployment fails for any reason. Sometimes, you can restart a suspended reorganization. Both procedures are explained in the following subsections.

Abandoning Instantiation
If the deployment fails at any step, you can abandon the instantiation.

```
jadclient path=c:\jade\system ini=c:\jade\jade.ini server=singleUser schema=RootSchema app=JadeReorgApp endjade action=abortReorg
```

Then unversion the changed schemas using JADE Loader.

```
jadloadb path=c:\jade\system ini=c:\jade\jade.ini server=singleUser unversionAllSchemas=true
```

If abandoning the instantiation fails, you can restore the database from the last full backup; either the backup from step 2, or a previous backup, in which case you could perform a roll-forward recovery.

Restarting the Reorganization
In this example, concurrent updates by other applications during the reorganization are disallowed (`reorgAllowUpdates` is set to `false`). Although the deployment is performed in single user mode so no other applications can be using the database, this option has the effect of automatically aborting the reorganization if an error occurs.

If you allow updates, an exception can occur resulting in the reorganization being suspended. For example:

- Transition cannot be initiated because a class is in use
- Lock exception occurs when the reorganization is committed

You must then manually restart the reorganization as follows (or abandon the instantiation as described at the start of this error handling section).

```
jadclient path=c:\jade\system ini=c:\jade\jade.ini server=singleUser schema=RootSchema app=JadeReorgApp endjade action=restartReorg
```

Scenario 2
Consider an application where high availability is required. It is acceptable for the time required to complete the deployment to be longer, provided the system remains available for as long as possible.

The developer would organize the schema and forms definition files to be deployed using a single `.mul` file as in Scenario 1.

1. Load the schema and forms files, suppressing any reorganization.

```
jadloadb path=c:\jade\system ini=c:\jade\jade.ini server=multiUser schemaFile=myDeploy.mul suppressReorg=true
```

The default `loadStyle` of `latestSchemaVersion` is assumed. The system remains available while the schema and forms files are loaded.

2. Initiate the online reorganization phase (up to the transition).
The system remains available while the online phase of the instantiation is performed. The database remains in dual update mode until the transition is initiated.

3. Shut down the system.

If possible, perform graceful system shut down (signoff users, and so on). Shut down the database server in order to stop server applications.

The system is no longer available until the instantiation completes.

4. Initiate the offline phase.

5. Restart the system. The system is now available to users.

Error Handling

If an error occurs before the reorganization has been initiated, you can abandon the schema instantiation, with the system remaining available, by aborting the reorganization.

Then unversion the changed schemas using JADE Loader.

Scenario 3

Consider a system that has a secondary database on which inquiry applications are running. To enable the transition to be scheduled on the secondary database (the transition occurs automatically in Scenario 2), a managed transition is initiated.

Secondary database tracking is halted at the end of the online phase prior to entering the offline phase so that the required downtime for inquiry applications can be scheduled.

1. Load the schema and forms files into the primary database, suppressing any reorganization.

The default loadStyle of latestSchemaVersion is assumed. The system remains available while the schema and forms files are loaded.

2. Initiate the online reorganization phase on the primary database (up to the transition).
The system remains available while the online phase of the instantiation is performed. The primary database remains in dual update mode until the transition is initiated.

The secondary database enters the **Seeking Approval** state; that is, the replaying is halted until the disposition of the reorganization on the primary is known. The secondary database remains available in this state.

3. Shut down the primary database.

If possible, perform graceful system shut down (signoff users, and so on). Shut down the database server in order to stop server applications.

The primary system is no longer available until the instantiation completes.

The secondary system remains available.

4. Initiate the offline phase on the primary system.

```
jadclient path=c:\jade\system ini=c:\jade\jade.ini server=singleUser
   schema=RootSchema app=JadeReorgApp endjade
   action=initiateManagedTransition
```

5. Restart the primary database, which is now available to users.

The secondary database replays 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.

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

### Scenario 4

Consider a very large system into which a new schema has been deployed. The schema has been versioned and the instantiation is expected to take some time. If a bug is detected in a method of the still-running application and needs to be patched, the change needs to be made to the current version of a method using the `currentSchemaVersion` load style. Any changed methods are activated as soon as the compile transaction is committed, which enables this load style to be used to affect the current runtime behavior of the application.

Load the schema and forms files, as follows.

```
jadloadb path=c:\jade\system ini=c:\jade\jade.ini server=multiUser
   schemaFile=hotfix3456.scm loadStyle=currentSchemaVersion
```
Summary

This paper has presented the features in JADE to facilitate schema evolution and schema instantiation.

The general concepts were defined and then both the general principles and details of schema evolution were discussed. After explaining how versioning was used to represent both the current and latest representation of the metaschema, details of the types of entities that are versioned and under what circumstances was presented.

The mechanisms available in the development environment to support working with multiple versions of a schema were described including the visual cues that indicate whether the developer is working in the current or latest schema context. Also presented were tools to investigate the entities that have been versioned.

The phases involved in instantiating any changes were then described and the online and offline phases were explained. After discussions of issues related to recovery, SDS, RPS, and a number of deployment scenarios were described in some detail.

The paper concludes with appendices describing some of the significant changes in these features in JADE 6.1 and guidelines on how to take full advantage of these features, and a glossary of technical terms.
Appendix A       JADE 6.1 Differences

In JADE 6.1, significant modifications were made in the areas of schema evolution and instantiation. The main motivation for these modifications was to significantly increase the availability of the system, both in development and in deployment. This appendix highlights some of the important differences in these features between JADE 6.1 and earlier releases.

In the JADE development environment, the changes enable objects to be accessed even if their specifications are being changed. Increasingly, systems run 24/7 and changes must be achieved in narrow time windows. The changes are designed to enable you to make changes and then deploy them into a running system with as much uptime as possible.

### Schema Evolution Changes

<table>
<thead>
<tr>
<th>JADE Releases Prior to 6.1</th>
<th>JADE 6.1 Release</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cannot change a class that is in use.</td>
<td>Class in use can be changed and versioned.</td>
</tr>
<tr>
<td>Cannot use a versioned class.</td>
<td>The current version of the versioned class can continue to be used.</td>
</tr>
<tr>
<td><strong>Schema</strong> never versioned.</td>
<td><strong>Schema</strong> versioned to provide current and latest schema contexts.</td>
</tr>
<tr>
<td>Only structural versioning of <strong>Class</strong>, <strong>Property</strong>, and entities related to the property change.</td>
<td>Structural and non-structural versioning performed so additional entities such as <strong>Method</strong>, <strong>Constant</strong>, <strong>Locale</strong>, <strong>Form</strong> are versioned to enable the current version of the system to run unchanged.</td>
</tr>
<tr>
<td>Versioning a class causes its subclasses and subschema types to be versioned. All properties of all these types are also versioned.</td>
<td>Only entities that must be versioned are versioned. Unchanged properties, subclasses, and subschema types are not versioned automatically when a class is versioned.</td>
</tr>
<tr>
<td>If an encrypted schema is loaded, no source is saved.</td>
<td>If an encrypted schema is loaded, an encrypted version of the source is saved unless the <strong>DontSaveSource</strong> option is used. This enables the compiler to decide if methods have changed and therefore should be versioned.</td>
</tr>
<tr>
<td>One load style but JADE Loader had a <strong>NoStructuralChanges</strong> option. After a system is versioned, the change must be instantiated to move forward.</td>
<td>Three load styles for greater control over schema load and the entities versioned. The latest version of the schema and any dependant schemas can be removed by unversioning the schema.</td>
</tr>
</tbody>
</table>
Development Environment Changes

<table>
<thead>
<tr>
<th>JADE Releases Prior to 6.1</th>
<th>JADE 6.1 Release</th>
</tr>
</thead>
<tbody>
<tr>
<td>Each schema appears once in the Schema Browser.</td>
<td>If a schema is versioned, it appears twice in the Schema Browser, to enable you to select the schema context in which you want to work.</td>
</tr>
<tr>
<td>No indication given that an entity is versioned.</td>
<td>Versioned entities indicated by color and icon overlays.</td>
</tr>
<tr>
<td>No facilities to find classes that are versioned.</td>
<td>Tools provided to display and compare versioned methods and other entities.</td>
</tr>
<tr>
<td>Could view or make changes only in the latest version of any class.</td>
<td>Can view and make changes in the current or latest schema context.</td>
</tr>
</tbody>
</table>

Schema Instantiation Changes

<table>
<thead>
<tr>
<th>JADE Releases Prior to 6.1</th>
<th>JADE 6.1 Release</th>
</tr>
</thead>
<tbody>
<tr>
<td>Must shut down all applications before beginning a schema load.</td>
<td>Schema load can be performed online.</td>
</tr>
<tr>
<td>Applications cannot update files involved in the reorganization.</td>
<td>Applications can access and update objects without restriction during the online phases of the reorganization.</td>
</tr>
<tr>
<td>Database file reorganization is single-threaded.</td>
<td>Multiple reorganization worker threads can be used to improve performance.</td>
</tr>
<tr>
<td>Transition to latest version happens automatically at the end of reorganization.</td>
<td>An administrator can control when the transition to the latest version is initiated.</td>
</tr>
<tr>
<td>File compaction possible only if system is offline or files are converted to read-only mode.</td>
<td>Can compact a file on a live database.</td>
</tr>
</tbody>
</table>
Appendix B       How to Work With These Features

Taking full advantage of the benefits of schema evolution and instantiation requires you to work slightly differently with JADE. There is no way to ‘disable’ these features and revert to the behavior in earlier JADE releases. The following sections offer guidelines for working with these features.

1. Choose the Correct Schema Context (Development)

Although you can version a schema manually, it happens automatically when a structural change is made to a class that has instances.

If you made the change, your schema context is automatically adjusted to be the latest schema version. If another developer made the change, you are informed of the change through a message box, but your schema context is not adjusted.

In most cases, you want to make changes that only affect the latest schema version. If you are working in the latest schema, your Class Browser is decorated, by default, with the Version Latest Gold JADE skin. Think of this as the Golden Rule (pun intended).

Occasionally, you want to make changes to a method or form without carrying out a schema instantiation. In that case, select the current schema context when your Class Browser is decorated, by default, with the Version Current Blue JADE skin.

When you log on, remember to select the appropriate schema context, which is usually the latest schema context.

Even if you make changes in the current schema context by mistake, in most cases they are propagated forward to the latest schema version. If the changes are not propagated forward, a dialog warns you of the situation.

You can prefer to see the current and latest version of attributes, constants, and methods at the same time, by selecting the View | Show Composite View menu command.

2. Schedule Schema Instantiations (Development)

You could instantiate the schema (reorganize the database) after each change that causes the schema to be versioned. This practice would reduce the number of schema contexts to one.

However, continually reorganizing the database simply to avoid having two schema contexts is rather extreme, and ignores the main benefit of schema evolution, which is that after you make a structural change, you do not have to instantiate the schema for applications to run. You can take advantage of this feature to schedule schema instantiations to suit the work patterns of your development team.

3. Use the Appropriate Load Style (Deployment)

In Scenario 1, the onlyStructuralVersioning load style is recommended for deployments where high availability is not essential but needs to be completed as quickly as possible with minimal administrative intervention. This type of completely offline deployment is virtually identical to the way schema loads were carried out in earlier versions of JADE.

In other situations, you can use the latestSchemaVersion load style to achieve maximum uptime when deploying changes to a system.
Glossary

**Current Version**
Original version of a metaschema entity that represents the existing state of user objects. User applications and **JadeScript** methods use the current version of a class.

**Data Pump Application**
Server application used by a Relational Population Service (RPS) node to implement incremental object replication into a target relational database.

**Database File Compaction**
Special form of file reorganization where objects are cloned but not mutated. Goals are to reclaim free space and eliminate file system fragmentation.

**Delta**
Change control mechanism for checking out and testing changes to methods. Changes that appear successful can be checked in; less successful changes can be discarded.

**Dual Update Mode**
Phase in schema instantiation when updates are applied to user objects in the current `.dat` file and to the mutated user objects in the `.reo` file.

**File Reorganization**
Phase in schema instantiation when objects are copied to a new `.reo` file. The objects are mutated, if necessary, to match the latest version of the class definition.

**File Synchronization**
Phase in schema instantiation when transactions that occurred during file reorganization are applied to objects in the target `.reo` file.

**Inter-Object Conversion**
Phase in schema instantiation when relationships between objects in the `.reo` file are established, including population or repopulation of dictionaries.

**Latest Version**
New version of a metaschema entity created as a result of a development change that has been made or loaded but has yet to be brought to life.

**Load Style**
Parameter to a schema load determining into which schema context changes are loaded and whether non-structural versioning is allowed.

**Metaschema**
Objects describing system entities such as schemas, classes, properties, and methods.

**Non-Structural Versioning**
Versioning to ensure that applications can run unchanged. For example, a method change loaded into the latest schema context leaves the current version unchanged.
Object Mutation
Phase in schema instantiation where an object’s structure is converted into a form compatible with the class definition in the latest schema version.

Patch Control
Assigning a numeric patch version number to changes to schema entities to enable you to track and extract changes by patch version.

Recovery
The two types of recovery are restart recovery, to automatically recover after a crash, and archival recovery, to roll forward through database journals after restoring a backup.

Relational Population Service
Specialized secondary database server node that implements incremental object replication into a target relational database.

Replayable Database Reorganization
Reorganization that is audited in the journals to support restart in case of interruption, roll-forward recovery, and replay on secondary databases.

Schema Context
Version of a schema (latest or current) selected in the development environment or for a schema load.

Schema Evolution
Changing the specifications of a system by adding, deleting, or changing entities such as schemas, classes, or properties.

Schema Instantiation
Bringing the changes introduced by schema evolution to life.

Secondary Database Service
Database server node that manages a database with a secondary database role.

Structural Versioning
Versioning to make the reorganization phases aware which objects require updating.

Transition
Final phase of schema instantiation when instances of versioned classes are no longer available to applications until the instantiation completes. Must be carried out offline.

Unversioning
Removal of the latest version of schemas to discard pending changes before those changes have been instantiated.

Versioning
Creation of metaschema objects to represent both states of the system (current version and latest version). Enables schema evolution to occur.