

ALBPM Project Versioning

A Guide to Understanding Project Versioning
ALBPM 5.5 and ALBPM 5.7

Updated: December 22, 2006

Table of Contents

Table of Contents	2
<i>Introduction</i>	3
<i>ALBPM Projects</i>	3
Project Versioning	3
Revision Changes – Compatible Processes	5
Minor and Major Version Changes – Incompatible Processes	6
Project Revision Changes – Compatible Project Catalog Change	7
Project Minor and Major Version Changes – Incompatible Catalog Changes	7
Instance serialization/de-serialization related problems	8
Using a catalogued database table or a business object that extends a catalogued database table as instance variable	8
Changing non-versionable jars	9
Changing the non-versionable external resource interface	9
Changing the versionable jars while preserving the process versions	10
Other conditions that should be prevented	10
<i>Project Consistency Check</i>	10
<i>Integrated External Resources</i>	10
<i>Summary</i>	11

Introduction

This document describes ALBPM project component versioning when published and deployed within a ALBPM Enterprise environment. Fuego uses project versioning to seamlessly enable the evolution of business processes as well as the cataloged components related to the process, while simultaneously controlling the myriad effects of having multiple concurrent versions in production.

A good understanding of how ALBPM manages versions for all project components is critical for a successful deployment and continued management of business processes implemented and orchestrated by ALBPM.

Throughout this document, we will refer to components with the new ALBPM 5.7 naming convention although the compatibility and incompatibility content remains the same for both ALBPM 5.5 and ALBPM 5.7.

ALBPM Projects

Fuego manages the concept of projects, which is simply an abstraction to encapsulate both the business processes and the components that, as a group, define a unit of deployment in an ALBPM Process Execution Engine. These two entities are grouped together to enforce consistency across the business process and integration layers. Therefore, when thinking of versioning, it is imperative to think in terms of project versioning.

Project Versioning

ALBPM has supported versioning at the business process level from the earliest releases. The notion of business process versioning enables the introduction of changes to existing business processes (i.e., new process versions) without disrupting the flow of existing instances in previously deployed business processes (i.e., old process versions).

With the introduction of the 5.X releases, Fuego introduced support for the versioning of projects. Because business processes rely on the components they invoke, the need to version more than just the process model was evident. Project versioning allows just this type of control.

Depending on the nature of the changes, ALBPM allows different levels of versioning for projects. ALBPM allows revision changes as well as minor and major version increments to define breaks in process compatibility. Each type of version is described in detail below.

Revision Versioning: By default, ALBPM will determine if a change qualifies as a revision, and if it does, ALBPM will propose to deploy the project as a revision to the existing deployment. Changes that qualify as revisions are listed in Figure 1.3. The changes in a revision apply to all new instances and the existing in-flight instances in the same version, since the changes are compatible with the previous version.

Minor and Major Versioning: Minor and Major versions are automatically proposed during deployment by ALBPM if business process or component compatibility is broken. Alternatively, during deployment one can manually select to increment to a new Minor or Major version if there is a desire for an explicit break point in versioning as the project evolves. This allows multiple versions of the process to be simultaneously executed; instances already in progress would continue to flow in their version, and new instances would follow the latest deployed version of the process.

Figure 1.1 illustrates the version numbers more clearly, as viewed in the ALBPM Process Administrator:

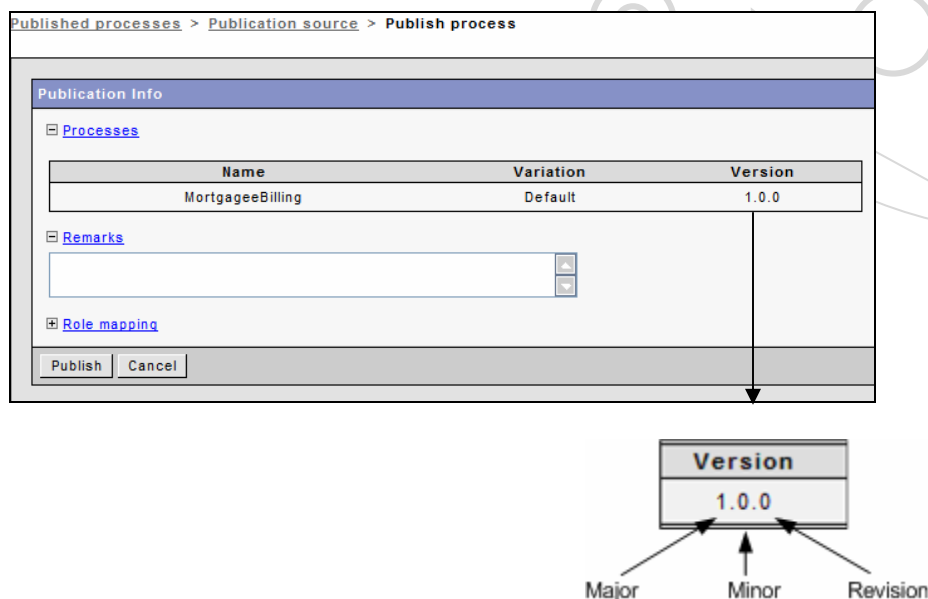


Figure 1.1 – Version number sequence

Figure 1.2 is an example of how deployed processes within a business environment translate to versions over the course of a year:

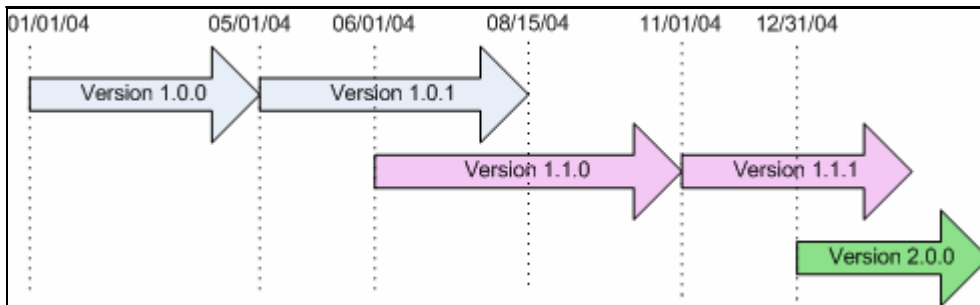


Figure 1.2 – Process Version Lifecycle

Note: The colored arrows represent instances flowing in a process. When new revisions are introduced, in-flight instances begin to use the newly revised process definition. When new major or minor versions are introduced, the new instances use the new definition while in-flight instances continue to use the older version (for example, from 06/01/04 through 08/15/04, two versions were in production with in-flight instances; in the example all the instances in version 1.0.1 reached the “End” by 08/15/04).

Revision Changes – Compatible Processes

This section outlines the changes that constitute “revision” versioning. These changes do not automatically force a minor or major version change during the publish phase.

- Adding a Role to a business process
- Deleting an empty Role (one with no activities in it) from a business process
- Adding an activity to a business process
- Adding a Global or Grab activity to a Role in a business process for a project
- Deleting a Global activity from a role in a business process
- Changing Fuego Business Language (FBL) business rules in a business process activity
- Changing Conditions or Due Intervals for an existing business process transition
- Adding any kind of transition to a business process
- Deleting a transition from a business process
- Adding an instance variable to a business process
- Deleting an instance variable from a business process
- Adding a variable to an argument mapping for a business process
- Deleting a variable from an argument mapping for a business process
- Changing the implementation type for a business process activity. For example from activity FBL to Procedure or Screenflow.
- Modifying a Fuego Screenflow (Adding to, deleting from or modifying its flow or activities)

- Modifying a Fuego Procedure (Adding to, deleting from or modifying its flow or activities)

Minor and Major Version Changes – Incompatible Processes

This section defines the reasons for which minor or major versioning is automatically forced.

- Deleting an activity from a business process. The removed activity may have instances waiting for execution in it; if this activity is removed, all the instances in this activity will be lost as well (this excludes Global activities because they do not have instances assigned to them)
- Changing the type of an instance variable in a business process; the change in the variable type may make existing values for business process instances incompatible with the current type
- Changing business process activity types; for example, changing from an interactive to an automatic activity type constitutes a major change in the generated Java classes since the execution models for these activity types are very different
- Moving an activity from a Role to a different one; all instances in the activity being moved must not be selected by a participant (it is responsibility of the process owner to ensure that all the instances are properly unassigned and then re-assigned to the right participants).

As a reference, Figure 1.3 shows the types of process changes that are both compatible and incompatible.

Type of Process Change	Compatible	Incompatible
Add a variable to an argument mapping	✓	
Delete a variable from an argument mapping	✓	
Change the implementation type for an activity	✓	
Delete a variable from an argument mapping	✓	
Add a role	✓	
Delete an empty Role (no activities in it)	✓	
Add an activity	✓	
Add a Global or Grab activity to a role	✓	
Delete a Global or Grab activity from a role	✓	
Change Business Rules in a business process activity	✓	
Change Condition or Due interval for a transition	✓	

Type of Process Change	Compatible	Incompatible
Add a transition	✓	
Delete transition	✓	
Add an instance variable	✓	
Delete an instance variable	✓	
Modify a procedure	✓	
Modify a screenflow	✓	
Delete an activity		✓
Change the type of an instance variable		✓
Change business process activity types		✓
Move an activity from a role to a different role		✓

Figure 1.3 – Table of compatibility with previously published process

Project Revision Changes – Compatible Project Catalog Change

The following changes are allowed to components cataloged in the ALBPM project catalog. After publishing and deploying these changes, in-flight instances will start using the changes implemented in this new project revision at the component level.

- Adding BPM Objects to the project's catalogue.
- Adding a new introspected component.
- Adding a new module to the catalog.
- Deleting or removing a BPM Object from the project's catalogue as far as this component is not the type of any process instance variables (process instance variable, external variable or business variable).
- Adding a new attribute to a BPM Object in the project's catalogue.
- Modifying a BPM Object presentation for a BPM Object in the project's catalog.
- Clone a component to another module.

Project Minor and Major Version Changes – Incompatible Catalog Changes

There are a series of changes that are not currently automatically identified by the project checker routine that can bring incompatibilities with in flight instances when a new revision is created for a project and not manually generating a minor or major project revision forcing the appropriate process versioning.

The following is supposed to be an extensive list of catalog changes that may make the process instance information incompatible.

Instance serialization/de-serialization related problems

Each business process implemented in a business process can have a set of variables defined to carry context as the instance moves through the process. The underlying implementation of these set of variables is aggregated in a Java Object that will have an attribute for each defined process instance variable. If the type of any of these instance variable changes (for example from String to Int), this action will not be detected by the project publication routine since it does not have knowledge as to whether there are in flight instances when this process is finally deployed to a Process Execution Engine. This will introduce a de-serialization problem for process instances already hosting values for these variables when the Process Execution Engine accesses this instances, as it is not always possible to accomplish an automatic casting from a “String” to an “Int” value. Similarly for any other instance variable type which includes complex objects with inner attributes. It would not be acceptable to change the type of an inner attribute of a complex object such as an attribute of a BPM Object. However, when the Process Execution Engine will try to de-serialize the instance created before the project revision was published and deployed, it will try to assign the “old” Business Object to the “new” Business Object, which may lead to an exception when the instance will try to access to that attribute . A similar situation could happen if a non-versionable catalogued Java object is an instance variable and this object suffers some structure modifications or whether its Serializable suffers modifications. For Serializable Java Objects, it is recommended that an explicit serial version is included as part of the class definition to prevent Java Serialization issues.

Possible solution to mitigate the risk

Avoid using non-versionable java packages if the Java Objects in the package are supposed to have frequent changes.

A best practice recommendation would be to keep the instance variable types as simple as possible whenever possible.

Avoid changing the attribute types for simple or complex objects. Add new attributes and deprecate the old ones instead.

Using a catalogued database table or a business object that extends a catalogued database table as instance variable

When a database table is used as a process instance variable, the table structure or metadata is stored along with the instance information serialized in the Process Execution Engine Database.

If the table structure changes (a column is deleted, a column is renamed or a column type changes) and the SQL component is re-catalogue or re-introspected, existing instances will have a metadata different from the one to

access the new database structure and there will be casting and incompatibility problems. For example, if you remove a column, the already stored instance information may produce a SQL query that will try to retrieve the removed columns producing an error at runtime.

The same condition applies to BPM Objects when these extend or inherit from SQL or Database components as they will also keep the underlying Database table structure along in the instance information.

Since the SQL table structure is stored in the instance, even if you re-catalogue the table, and recreate the business object that extends the table, the query will still use the old structure and generate a runtime error.

Possible solution to mitigate the risk

The problem happens only if a table or database field/column is removed, renamed or the column type changes. To solve the problem, avoid removing tables, columns or changing the column types and follow a deprecation model.

Changing non-versionable jars

When you modify a non-versionable jar in your project catalog, the instances from the old process version will start using that jar. If a method from a class or interface was eliminated or its signature changed, you may experience a `NoSuchMethodException` when trying to execute a non existing method.

Depending on the changes in the classes contained in a jar file, it is possible to get different exception types. For example, if a new attribute was added to a class, you may experience a `NullPointerException` since no value exists in the existing in flight instances.

Please note that this could also happen even when you generate a new project version.

Possible solution to mitigate risk

Do not change non-versionable jars. If the jar will suffer frequent changes, it is recommended to catalog the jar as a versionable one.

Changing the non-versionable external resource interface

Let's suppose that the project interacts with a legacy system.

There is some interface or protocol that regulates the calls to that system and the format of the messages traveling between ALBPM and the legacy system.

Now, suppose that by some reason the interface or protocol was changed.

You may have compatibility issues with the old instances using the new protocol or interface.

Possible solution to mitigate risk

While ALBPM can version processes and business objects, most of the external resources cannot be versioned. Keep the interfaces and protocols unchanged as much as possible.

Changing the versionable jars while preserving the process versions

ALBPM allows preserving the process version even if a incompatible change in a catalogued versionable jar was introduced. This implies that the developer potentially can change the signature of some method, making the new jar incompatible for in flight instances.

Possible solution to mitigate risk

Be extremely careful when preserving the process version. Increase the process version manually in case of doubt.

Other conditions that should be prevented

There are other catalog structural changes that may affect the process instance de-serialization producing runtime errors when accessing in flight process instances. The following is a list of these changes that should be avoided whenever possible:

- Deletion of Project Catalog Modules that are being used in process instance variables.
- Renaming of Project Catalog Modules that are being used in a process instance variables.
- Move components or objects from one module to another module in the project catalog.
- Change inheritance type for a BPM Object.

Project Consistency Check

Before running the internal compatibility algorithm, ALBPM automatically performs a consistency check. The reason for this consistency check is to enforce that the business processes, components and business rules are all correctly in place so that the changes implemented in the different project areas can be successfully published and deployed.

It is recommended to perform a consistency check periodically from the ALBPM Studio so that errors are not found at project publish time.

Integrated External Resources

Thus far, only the versioning behavior for elements managed within the ALBPM Studio project boundaries has been explained. There are resources outside the project boundaries that cannot be considered by the automatic versioning algorithm implemented within ALBPM. It is very important to synchronize these external resources to avoid inconsistencies and failures when invoking them. These resources include:

- **JSPs, HTML, etc:** It is important to define a synchronization strategy for these resources since modifications in a JSP, or any other HTML rendering technology, may also require changes in the way it is invoked for it to work correctly.
- **RDBMS table and stored procedure changes:** When changes are made to a table or stored procedure accessed by ALBPM, the external resource definition must be manually refreshed or it may cause runtime problems.
- **Client/Server components:** If the interface to access a server component changes, the client side will need to change as well. For this reason, it is very important to define a synchronization strategy so that changes made to the server side components are also made to the client side at the same time to avoid inconsistency and runtime problems.

Summary

In summary, the revision, minor and major versioning is decided at publish time. When publishing a project, ALBPM will always try to revision first. If project compatibility is lost due to some of the reasons outlined above, ALBPM will automatically propose a new minor project version during publishing. Catalog changes are considered compatible and will generate revisions, but certain conditions must be considered to determine if a manual change to the version is a better choice.

If a new minor or major project version is desired, this can be manually accomplished by selecting the minor/major version increment option in the ALBPM Process Administrator.

In order to reduce the risk of process failure, it is also very important to consider all integrated external resources when making changes.