Using the DBUpgrader Plugin

# Overview

This KB article is intended to provide some guidance on how to use the DBUpgrader plugin. It will show how to setup a basic process, the common syntax of the XML files used and finally, provide some advice about how to avoid the “gotchas” which can occur with this plugin.

# Introduction

The DBUpgrader plugin is intended to allow you to specify steps within a process which will allow you to run controlled SQL against a database and if it fails, to have the option of running an associated rollback script. The plugin works by using a table within the database that tracks versions of SQL that have been deployed before and therefore giving you to option to rollback to specific points or rollback everything.

The plugin provides two steps. The first one is associated with deploying “UpgradeDB” SQL, whilst the second one is used for “RollbackDB” SQL. Each of the steps has the same sort of parameters associated with it which allow you to specify the target database you want to connect to, the JDBC connection information and the names of the XML scripts that you want to run. You can use these steps however you like when you design your deployment process. More information on this setup and the syntax of the XML files is covered below.

# Table Definition

The table used to track the status of the upgrade is usually created within the first XML upgrade file that is deployed and looks similar to the following:

CREATE TABLE APP\_VERSION

(

app\_name VARCHAR(255) NOT NULL,

app\_ver NUMERIC DEFAULT 0 NOT NULL

);

The table and field names used are at the user’s discretion, however these names should be consistent with the SQL you specified in the plugin to manipulate version information.

The table has two main columns: -

* APP\_NAME – This refers to the version of the application you are upgrading and is a free-text field. The value used comes from the XML <change-set release=”…”> element.
* APP\_VER – This refers to the change-set version which has been run so far. The value used here comes from the <change number="…"> element.

As each XML file gets processed, the APP\_NAME and APP\_VER will get updated (i.e. inserted) accordingly. This table will record what has been processed.

# XML File Syntax

The XML file used to drive the upgrade has several main components which are described below.

## The Header

The header is simply an element which describes the version of the XML being used and is standard to all XML documents, e.g.

<?xml version="1.0"?>

## The Change-Set

The change-set element is the main container used within the XML file that lists all the changes that are to be run. This element also contains an attribute which is used to describe the release version that this change-set is relevant for. This information gets inserted into the tracker table, described previously.

This change set element contains the following:

<change-set release="…">

<change number="…">

</change>

</change-set>

A change-set can contain 1..N <change> elements.

## The Change(s)

The change element is what is used within the XML file to describe the SQL that should be run against the target database. These change elements are numbered and the version associated with them is stored in the table described above.

This change element contains the following:

<change number="…">

<description>

Free text describing what you are doing

</description>

<sql separator=";">

DDL or DML SQL Statements that define what to upgrade

</sql>

<rollback>

<sql separator=";">

DDL or DML SQL Statements that define how to rollback

</sql>

</rollback>

</change>

The SQL specified can be any ANSI standard SQL that the specified JDBC driver can process. These SQL statements however must be separated by a “;” to run.

Example SQL:

<change number="2">

<description>

Insert more component data.

</description>

<sql separator=";">

<![CDATA[

UPDATE components\_reg SET superceded = 'N';

INSERT INTO components\_reg

(component\_id, component\_name, create\_date,

update\_date,description,superceded)

VALUES (doc\_seq.nextval,'CMP\_TS3',SYSDATE,NULL,NULL,'Y');

INSERT INTO components\_reg

(component\_id, component\_name, create\_date,

update\_date, description, superceded)

VALUES (doc\_seq.nextval,'CMP\_TS4',SYSDATE,NULL,NULL,'N');

]]>

</sql>

<rollback>

<sql separator=";">

<![CDATA[

DELETE FROM components\_reg

WHERE component\_name IN ('CMP\_TS3','CMP\_TS4');

UPDATE components\_reg SET superceded = NULL;

]]>

</sql>

</rollback>

</change>

Note: CDATA should be used when specifying DML, but not DDL.

Examples

A set of example files are available to accompany this KB article on request.

# Defining the Process

The following is an ***example*** process on how to setup the plugin. It is provided as a sample only and is not intended to show best practice. It will however describe what the fields mean and how an initial process can be constructed.

## Application Process Definition

The application process to install the database upgrade is a very simple one and simply invokes a component install called “Install DB” as is shown in figure 1 below. This installs a component called “DbUpgradeApp”.

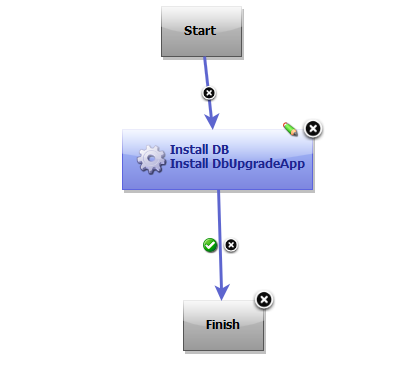


Figure 1- Application install process

## Component Process Definition

The component “DbUpgradeApp” is setup as a basic file system component and stores versions of the XML scripts that are used to drive the upgrade process. The overall component process that is invoked for this component is setup to comprise of the following: the first step downloads the artefacts controlled by the component, the second step is a shell command performs a DIR /S to show the current location on disc, the third step invokes the DBUpgrader Upgrade step to deploy the upgrade XML and the final step is only invoked on a failure to rollback the upgrade.

Figure 2 below shows this as laid out in the process designer.

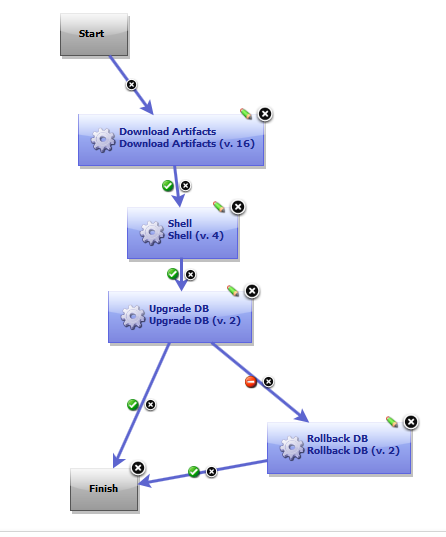


Figure 2- Component process

The properties for the Upgrade DB step are shown in figures 3 and 4 below. These give the database to connect to, the XML files to run and the SQL to query the versions already deployed.

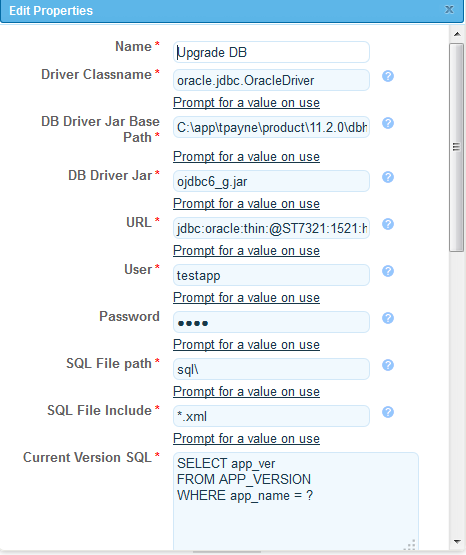


Figure 3- UpgradeDB properties

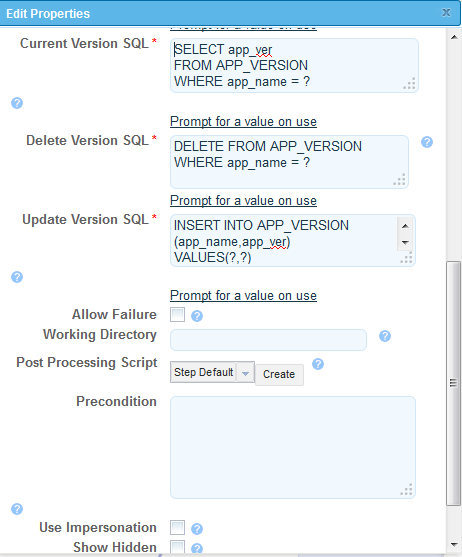


Figure 4- UpgradeDB properties (additional)

This plugin step is configured to run all \*.XML files in the sql/ sub-directory whenever the deployment is invoked. Obviously, this is not a great idea if you have files relating to many versions and applications, but you get the general idea.

The properties of the rollback step are shown in figures 5 and 6. They are virtually the same as the upgrade step above.

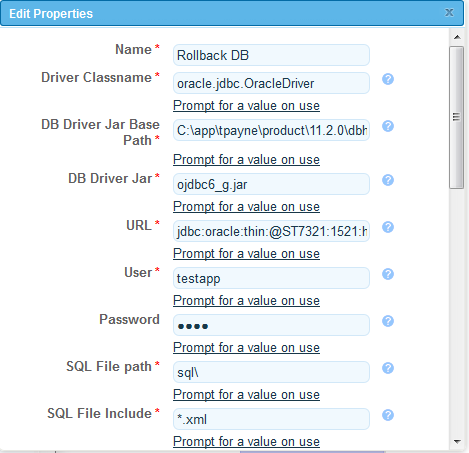


Figure 5- Rollback properties

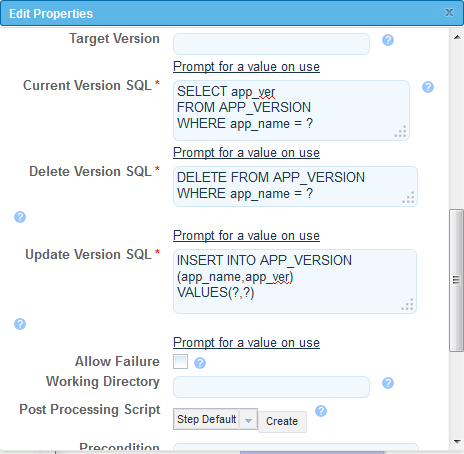


Figure 6- Rollback properties (additional)

Again, this plugin step is configured to run all \*.XML files in the sql/ sub-directory whenever the deployment is invoked.

# Some “Gotchas” Advice

The following is some small pieces of advice as to what to look out for when using this plugin:-

* Be careful of the sequencing used in <change-set release=”…”>. If one uses 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9, 1.10 for example, then this will run 1.1, 1.10, 1.2 etc., which may produce unpredictable sequences. Sorting is done using standard algorithms and which may produce the result described.
* Filter the XML correctly. The plugin will load all XML files specified in a directory and try to process them. This means that if the same directory is used for all XML upgrade scripts from the beginning, then if an error occurs, the rollback will be activate for ALL those XML files – back to the beginning. Therefore only load the XML files that need to be processed during the deployment – not everything. This can be done using the SQL file include.
* Test the deployment logic and rollback process first before using it in production. Also check the rollback performs as expected. As the process may rollback too much or too little.
* The rollback and deployment SQL does not support conditional logic. It also does not support languages like PL/SQL.