

# Provision and deploy database applications in various environments

## Introduction

---

This tutorial describes defining a process to perform application deployments for applications that use DB2 for z/OS, including schema changes. The deployment process uses the following software products:

- IBM UrbanCode Deploy (UCD)
- IBM Admin and Object Compare Tool for DB2 for z/OS
- z/OS Management Facility for z/OS Version 2.2

The deployment process includes steps for the following activities:

- Database schema provision
- Database schema changes
- Binding or rebinding packages for application changes
- Commands
- Utilities

## Before you begin

---

Ensure that the following prerequisites are ready:

- IBM UrbanCode Deploy Version 6.2 server, agent, and toolkit are installed correctly
- IBM z/OS Management Facility Version 2.2 server is installed correctly
- Source-code management client tools such RTC tools or Git tools are installed on the UCD server
- DB2 Administration Tool and Object Compare Tool Version 11.2 with [APAR PI67731](#)
- JDBC driver and RACF PassTicket jar files are available on the Unix Service System on your target subsystem. Consult your administrator for more information

- The following plugins are downloaded from IBM UrbanCode Deploy Plugin installed:
  - [File Utils plugin](#)
  - [SQL-JDBC plugin](#)
  - [zOS Utility plugin](#)
  - [z/OSMF plugin](#)
  - [Rational Team Concert - SCM plugin](#) if you use RTC as the source-code management (SCM)
  - [Git plugin](#) if you use GIT for SCM
- Proper access to UCD, z/OSMF, and SCM, and read and execution privileges for the files and folders from [APAR PI67731](#)

## Scenarios

---

This tutorial discusses steps for solutions for the following scenarios:

### [Deploying a test environment](#)

Developers or testers want a service to deploy a test environment with a database schema like another environment. The service can be invoked by a command or by any program or script.

### [Deploying application and database schema changes](#)

Developers or testers want a simple service that deploys checked-in application and database schema changes to target environments that they are authorized to deploy to. The database schema change deployment may involve complicated database operations including DDL, commands, and utilities. This scenario exploits IBM Admin and Object Compare Tool to compare the new schema to the target environment to create the process required to convert the target environment to the checked in schema version

### [Reviewing changes and the deployment process](#)

Administrators, including DBAs, security administrators, storage administrators, and others might need to review and approve the changes and deployment process in a controlled environment before or during the deployment.

## Binding or rebinding an application

Administrators or developers have an application that requires binding a new plan, or rebinding an existing plan to change the package list. The application might require new packages or package versions to be bound in the target environment. The packages might use different bind options. You might already have BIND cards defined for the packages to be bound.

This tutorial describes how to build deployment processes to address the above scenarios. It also talks about how to integrate the services with different SCM systems and how to automate the entire solution.

The services use existing plugins in IBM UrbanCode Deploy, and the z/OSMF workflows of IBM DB2 Administration Tool and Object Compare Tool. All of the artifacts used and files referenced in this tutorial are available in the template package that you can download.

## Implement with IBM UrbanCode Deploy

---

The sample template in the tutorial consists of:

- The components containing the artifacts
- The processes that prepare the artifacts and drive the provision and deployment
- The environments where the deployment will be run
- The application which associates the components and environments

The tutorial leads you through the following steps to build the two flows:

Step 1: Import the template

Step 2: Configure the components to associate with SCM systems

Step 3: Customize the process

Step 4: Add target environments and corresponding environment property files

Step 5: Setup the deployment style and notification

Step 6: Validate the process

## Create the solution

---

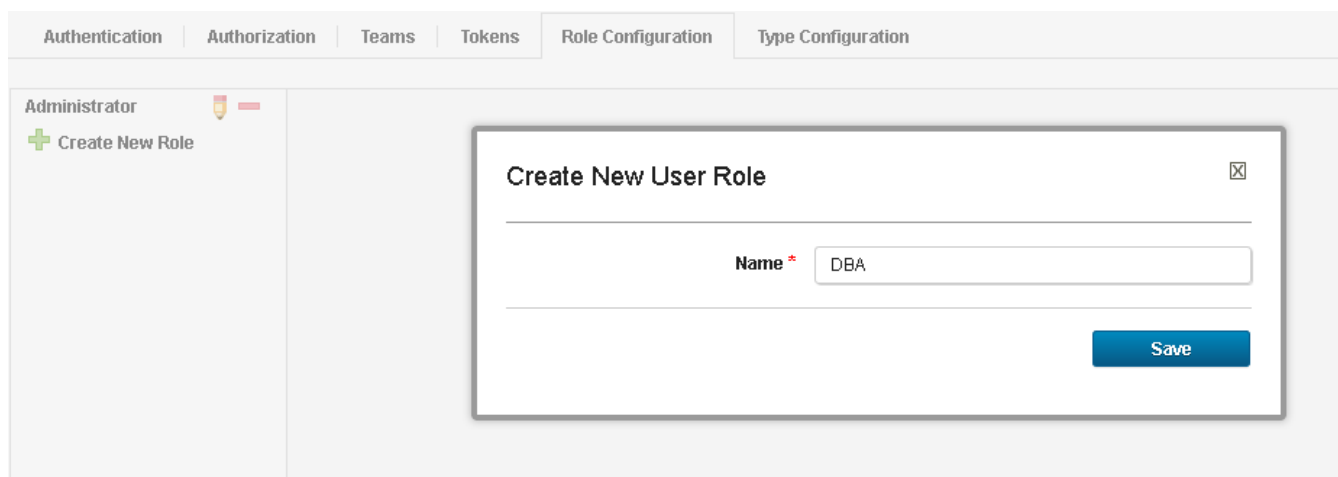
All steps described in this section are completed in the IBM UrbanCode Deploy web interface.

## Part I. Provisioning and Deployment Processes

### Step 1. Import the application

Step 1.a. Add a role named "DBA". Go to the Settings page, click **Security > Role Configuration**. Then click **Create New Role** on the left, enter `DBA`, and save the role. This role is assigned to user IDs for database administrators who can review and approve the schema changes to a controlled environment. The role name can be changed later, or if you already have a role defined for the purpose, you can drop it after you import the application. The tutorial describes about how to set and change the role name later.

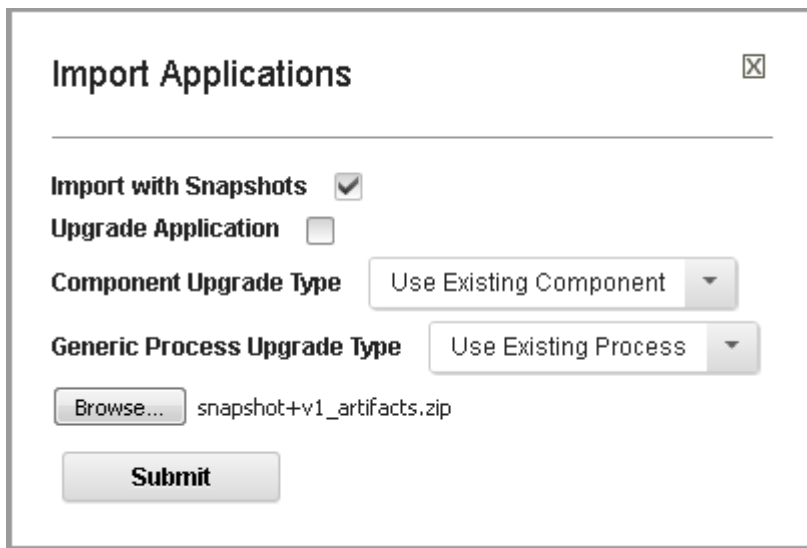
Figure 1. Create a New Role



The screenshot shows a web application interface with a top navigation bar containing tabs: Authentication, Authorization, Teams, Tokens, Role Configuration, and Type Configuration. The 'Role Configuration' tab is active. On the left sidebar, under the 'Administrator' user profile, there is a '+ Create New Role' button. A modal dialog box titled 'Create New User Role' is open in the center. It has a close button (X) in the top right corner. Inside the dialog, there is a text input field labeled 'Name \*' with the value 'DBA' entered. At the bottom right of the dialog is a blue 'Save' button.

Step 1.b. Go to the Applications page. Click **Import Applications**, and select **Import with Snapshots**.

Figure 2. Import Applications



**Import Applications** [X]

---

**Import with Snapshots** ☒

**Upgrade Application** ☐

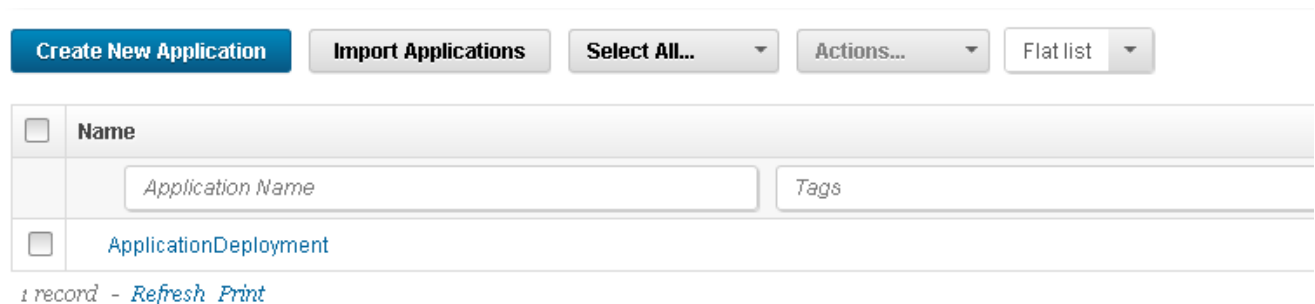
**Component Upgrade Type** Use Existing Component ▼

**Generic Process Upgrade Type** Use Existing Process ▼

snapshot+v1\_artifacts.zip

Click **Browse** and choose the downloaded template package file . Click **Submit** to import the application. You can now see the new application named ApplicationDeployment on the Applications page.

Figure 3. New application in the List



<input type="checkbox"/>	Name
<input type="checkbox"/>	<div>Application Name</div> <div>Tags</div>
<input type="checkbox"/>	ApplicationDeployment

1 record - [Refresh](#) [Print](#)

## Step 2. Configure the components

Step 2.a. Go to the Components page, and select the option for your SCM system:

- For Git, select the AppDeploywithAOCGIT component
- For RTC, select the AppDeploywithAOCRTC component

If you use other types of SCM systems, see [Creating components from source-code management systems \(IBM Knowledge Center\)](#).

Step 2.b. Go to the Configuration tab under the component and fill in the SCM system information.

Refer to Figure 4 for an example of customizing the Version Source Configuration section with your own Git system information. Ensure that Git is installed and is executable by UCD server, so that UCD server can communicate with the Git server.

The types of files in **Includes** are just samples in this tutorial, and you can fill in the types that your deployment requires.

Figure 4. GIT SCM as a sample

The screenshot displays the 'Basic Settings' page for a component configuration. The page is divided into a left sidebar with navigation links and a main content area. The navigation links include 'Basic Settings' (selected), 'Component Properties', 'Environment Property Definitions', 'Resource Property Definitions', 'Version Property Definitions', 'Configuration File Templates', and 'Version Import History'. The main content area is titled 'Basic Settings' and contains the following fields:

- Name \***: AppDeploywithAOCGIT
- Description**: Deploy app by running object compare w/o interrupt
- Teams**: DBATeam x SECADMTeam x +
- Component Template**: None
- Component Type**: Standard
- Version Source Configuration**:
  - Source Configuration Type**: Git
  - Repository URL \***: http://vm902103.svl.ibm.com:9090/kren/deploydemo
  - Branch \***: kren
  - Username**: kren
  - Password**: \*\*\*\*
  - Watch for tags**: ☐
  - Includes**: \*.txt, \*.json, \*.sh, \*.ddl
  - Excludes**: (empty)
  - GIT Path \***: git

Refer to Figure 5 for an example of customizing the Version Source Configuration section with your own RTC system information. Ensure that RTC SCM tools is installed and is executable by the UCD server, so that the UCD server can communicate with the RTC server. The file types in **Includes** are just samples in this tutorial, and you can fill in the types that your deployment requires.

Figure 5. RTC SCM as a sample

Dashboard
Usage
Configuration
Calendar
Versions
Processes
Changes

Basic Settings
Component Properties
Environment Property Definitions
Resource Property Definitions
Version Property Definitions
Configuration File Templates
Version Import History

### Basic Settings

Name \*

AppDeploywithAOCRTC

Description

Deploy app by running object compare w/o interrupt

Teams

DBATeam x SECADMTTeam x +

Component Template

None

Component Type

Standard

Version Source Configuration

Source Configuration Type

RTC SCM

RTC Server URL \*

https://qtdev.canlab.ibm.com:9443/ccm/

RTC Username \*

kren

RTC Password

.....

Stream \*

krenDevDeploydemoStream

Version Naming Convention

ID

Includes

deploydemoRTC/\*.txt

deploydemoRTC/\*.ddl

deploydemoRTC/\*.sh

Excludes

Include Root


☐

Command Path \*

/opt/rtcscm/jazz/scmtools/eclipse/scm

Step 2.c. Go to the Versions page under the component and click **Import New Versions** to validate the SCM settings. A new version is shown in the list if the configuration is correct. You might need to click **Refresh** under the list table to refresh the content.

Figure 6. Import a new version

Dashboard	Usage	 Configuration	Calendar	Versions	Processes	Changes
-----------	-------	---	----------	----------	-----------	---------

Import New Versions

Version	Statuses
<input type="text"/>	<input type="text" value="Statuses"/>

### Step 3. Customize the processes

The processes in the template package provide the capabilities to accomplish the following tasks:

1. Provision a schema like a source environment on a target environment with or without intervention
2. Deploy a schema change to a target environment with or without intervention

Intervention can be a required at this stage for review and approval by a certain role, such as a DBA or Security Administrator, before the changes are completed in a controlled environment; or the intervention can also be a pause by intention at a certain point to guarantee the changes to fit into a change window; or other proper reasons based on the routine protocol in your shops. The tutorial discusses customization of the process without intervention and the additional steps of the process with intervention.

Step 3.a. Go to the Configuration page under the component that you chose in Step 2, click **Component Properties**, and review and customize the properties.

*Figure 7. Component properties*

Basic Settings

Component Properties

Environment Property Definitions

Resource Property Definitions

Version Property Definitions

Configuration File Templates

Version Import History

Component Properties

Version 4 of 4

◀◀ ◀ ▶ ▶▶

Add PropertyBatch Edit

Name	Value
<input type="text"/>	<input type="text"/>
analyzeReportCard	REPORT
analyzeStep	cmban
compareStep	cmbcmp
compareStepJson	comparestep.json
compareWorkflow	compareworkflow.txt
deployReportCard	ADBRPTSM
deployStep	cmbm
genDDLWorkflow	genddlworkflow.txt
projectname	deploydemoRTC
runCompareWorkflow	runcompareworkflow.txt
runGenDDLWorkflow	rungenddlworkflow.txt
stepRegistry	stepregistry.txt
workingVarFile	workingVarFile.txt

Step 3.b. Go to the Processes page under the component that you chose in Step 2.

Figure 8. Process list under component

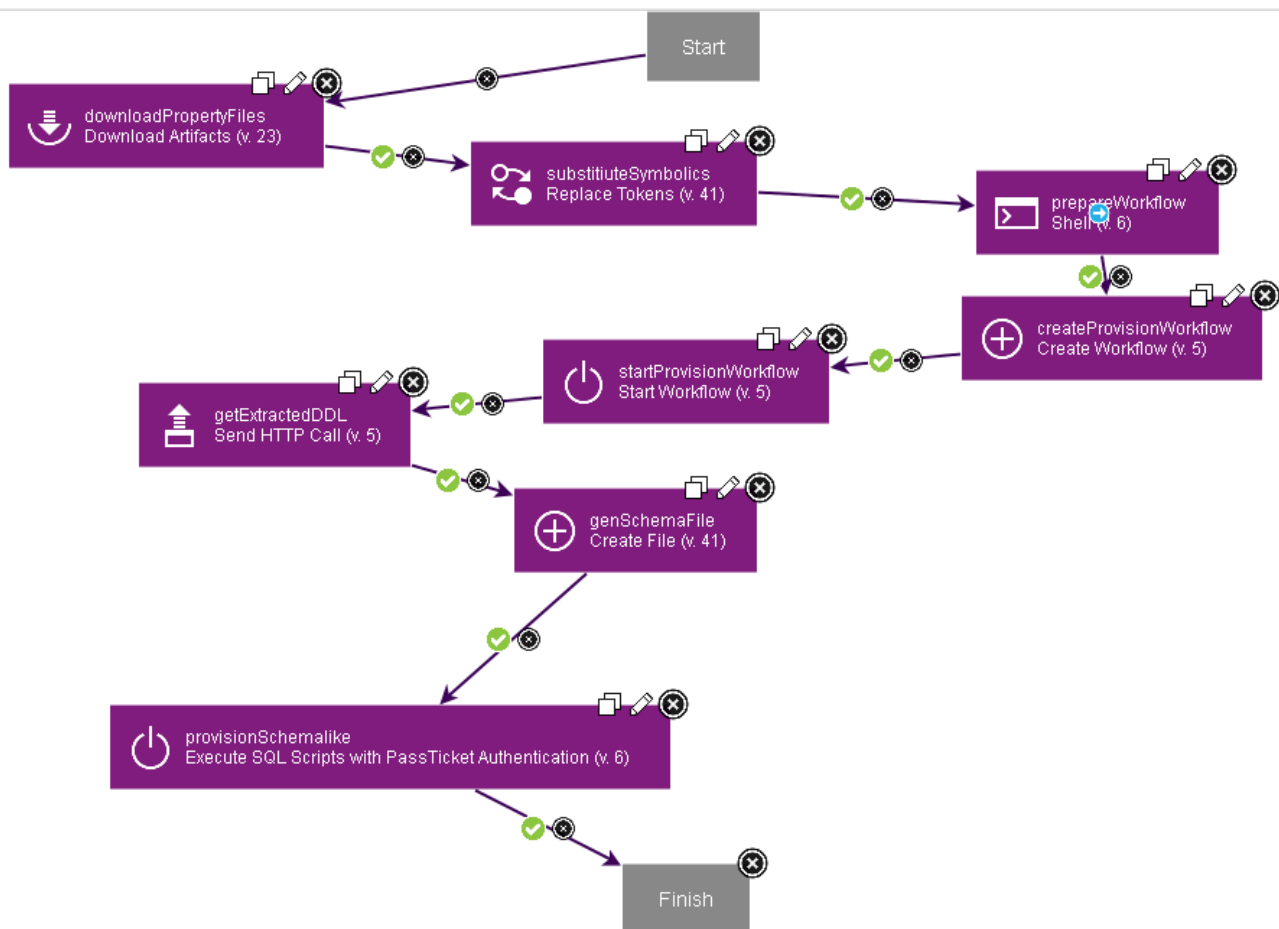
Dashboard	Usage	Configuration	Calendar	Versions	Processes	Changes
-----------	-------	---------------	----------	----------	-----------	---------

Create Process

Process
ProvisionLikewithApproval
ProvisionLikewithoutApproval
UpgradeApplicationSchemawithApproval
UpgradeApplicationSchemawithoutApproval

Step 3.b. Click **Edit** on the right side of the ProvisionLikewithoutApproval process. The view that is shown in the following figure opens.

Figure 9. Process ProvisionLikewithoutApproval



Step 3.c. The downloadPropertyFiles step downloads the required artifacts to the target environment. Open the step and customize the properties, then save the changes. Refer to Figure 10 if you are under the AppDeploymentwithAOCRTC

component and Figure 11 if you are under the component AppDeploymentwithAOCGIT component. The only difference is **Artifact Directory Offset**, where the process for RTC has project name.

Figure 10. Properties of downloadPropertyFiles of the process for RTC

## Edit Properties

---

Name *	<input type="text" value="downloadPropertyFiles"/>
Directory Offset *	<input type="text" value="."/>
Artifact Directory Offset	<input type="text" value="\${p:component/projectname}"/>
Includes *	<div><div>**/*</div><div></div></div>
Excludes	<div><div></div><div></div></div>
Sync Mode	<div>Sync ▾</div>
Full Verification	<input checked="" type="checkbox"/>
Set File Execute Bits	<input type="checkbox"/>
Verify File Integrity	<input type="checkbox"/>
Charset	<input type="text"/>
Working Directory	<input type="text" value="\${p:environment/srcDirectory}"/>
Post Processing Script	<div><div>Step Default ▾</div><div>New</div></div>

Figure 11. Properties of downloadPropertyFiles of the process for Git

## Edit Properties



---

Name *	<input type="text" value="downloadPropertyFiles"/>
Directory Offset *	<input type="text" value="."/>
Artifact Directory Offset	<input type="text"/>
Includes *	<div><div>**/*</div><div></div></div>
Excludes	<div><div></div><div></div></div>
Sync Mode	<div>Sync ▾</div>
Full Verification	<input checked="" type="checkbox"/>
Set File Execute Bits	<input type="checkbox"/>
Verify File Integrity	<input type="checkbox"/>
Charset	<input type="text"/>
Working Directory	<input type="text" value="\${p:environment/srcDirectory}"/>
Post Processing Script	<div>Step Default ▾ New</div>

You can choose to download certain types of files exclude others in the step. The environment-level property used in this step is covered later.

Step 3.d. Open the substituteSymbolics step, customize the properties and save the changes. You can use the **Property List** to replace the tokens in the downloaded artifacts with the key-value pair that is defined at environment level. This is useful when you have any environment-specific values to overwrite the input variables to the underlying tools that the process invokes. Alternatively, you can use **Property File Name** to define a textual property file with key-value pairs to replace the tokens in the download artifacts.

Figure 12. substituteSymbolics

## Edit Properties

<b>Name *</b>	<input type="text" value="substituteSymbolics"/>
<b>Directory Offset</b>	<input type="text"/>
<b>Include Files *</b>	<div>***</div>
<b>Exclude Files</b>	<div></div>
<b>Start Token Delimiter</b>	<input type="text" value="@"/>
<b>End Token Delimiter</b>	<input type="text" value="@"/>
<b>Property Prefix</b>	<input type="text"/>
<b>Property File Name *</b>	<input type="text" value="replace_tokens.properties"/>
<b>Property List</b>	<input type="text" value="\${p:environment/allProperties}"/>
<b>Explicit Tokens</b>	<div></div>
<b>Working Directory</b>	<input type="text" value="\${p:environment/srcDirectory}"/>
	Step Default ▼

Step 3.e. Open the prepareWorkflow step, customize the properties if needed, and save the changes. The script defined in the step converts the download artifacts to UTF-8 encoding to invoke the workflow in the following steps. This step is only needed if you don't have environment-specific values to overwrite the properties in the input variable file. If you do have environment specific values to overwrite the values in step 3.d, then you don't need this step. The encoding is converted automatically during the token replacement.

Step 3.f. Open the createProvisionWorkflow step, customize the properties and save the changes. The environment-level properties are covered later.

Step 3.g. Open the startProvisionWorkflow step, customize the properties and save the changes. The environment-level properties are covered later.

Step 3.h. Open the getExtractedDDL step, customize the properties and save the changes. For **Password**, enter `${p:environment/workflowPwd}`. The password is defined at the environment level later. This step gets the extracted data definition statement content from the source environment, and the next step stores the data definition statement content into a file that can be saved in SCM and reused in other rounds of provision, or reviewed by DBAs in the approval process.

Figure 13. getExtractedDDL

## Edit Properties ✕

---

Name \*

getExtractedDDL

URL \*

https://\${p:environment/WORKFLOWSERVER}/zosn

HTTP Method

GET ▾

Headers

Data / Data File

Request Content Type

JSON ▾

Allow Untrusted SSL Certs

☒

Username

\${p:environment/WORKFLOWUSER}

Password

••••

Output File

Working Directory

\${p:environment/srcDirectory}

Post Processing Script

Step Default ▾

New

Step 3.i. Open the genSchemaFile step, customize the properties if need, and save the changes.

Step 3.j. Open the provisionSchemalike step, customize the properties if need, and save the changes. The resource-level properties are covered later.

All the steps are now reviewed and customized for the provisioning process without approval.

If an intervention is required, for example, review and approval from DBAs before the process provisions the target environment, a review and approval step can be added.

Step 3.k. Go to the Processes page under the component. Click **Edit** on the right for the ProvisionLikewithApproval process. A Manual Task step is added right before the provisionSchemaLike step. Open the Manual Task step. This is to send the extracted data definition statement content to the DBA role, which was created in step 1, for review and approval. Customize the properties if you have different role name.

Figure 14. Manual task properties

Edit Properties

Name \* DBARReviewandApproval

Notification Template \* ApprovalCreated

Add Property

Name	Label	Pattern	Required	Default Value	Description	Actions
schemaContent	Schema Content		false	https://[p:environment/WORKFLOWSERVER]/zosmf/restfiles/ds/[p:environment/ddlOutputDataset]		Edit Delete

1 record - Refresh Print

Who can approve this task

Role Member By Component

DBA for Standard Component Remove

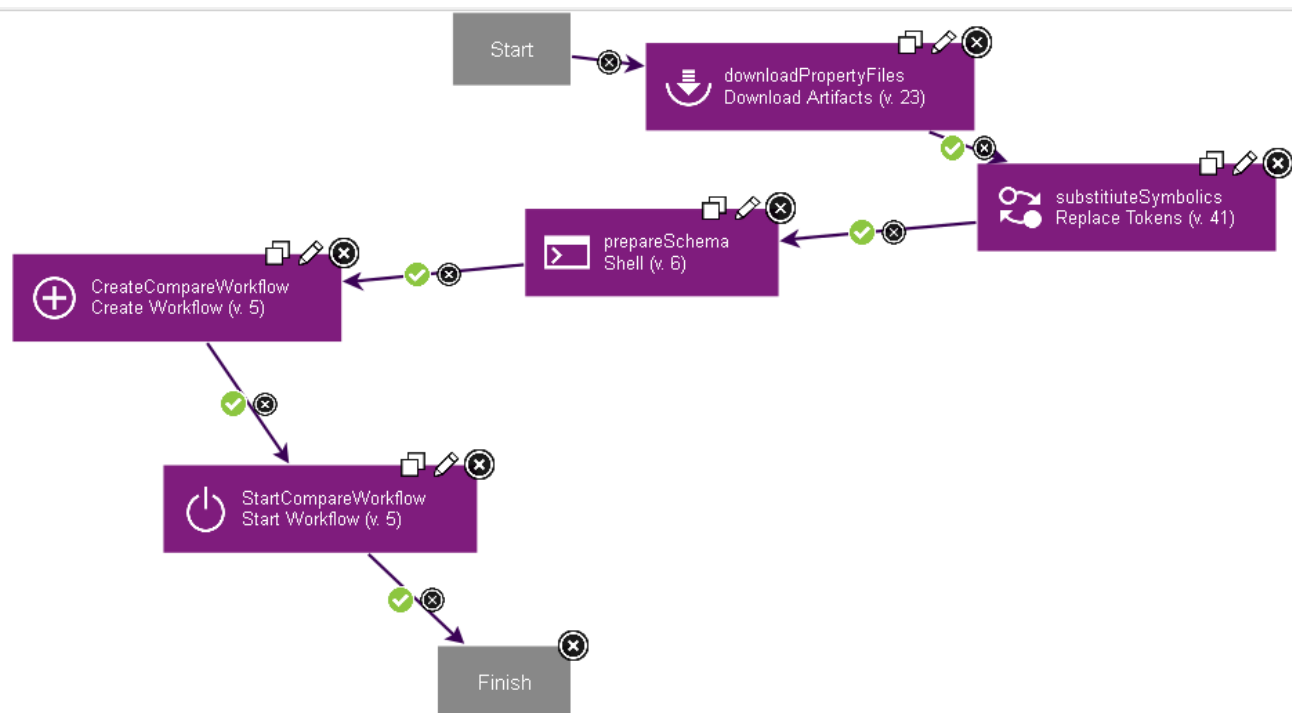
Add Role...

We now covered the process of provisioning schema like a source environment in a target environment.

When the schema changes along with the application change, the target environments must be upgraded to the new level. The UpgradeApplicationSchemawithoutApproval and UpgradeApplicationSchemawithApproval processes complete this action. Let's look at the UpgradeApplicationSchemawithoutApproval process first.

Step 3.l. Go to the Process page under the component. Click **Edit** on the right for the UpgradeApplicationSchemawithoutApproval process.

Figure 15. UpgradeApplicationSchemawithoutApproval process



The steps are very similar to the first 5 steps in the ProvisionLikewithoutApproval process and the ProvisionLikewithApproval process. Open each step and customize the values for the properties based on your environment. The environment-level, component-level, and resource-level properties are covered later.

That is so much for the UpgradeApplicationSchemawithoutApproval process.

In some shops, you might want to pause after the compare is done but before applying the delta changes to the target environment. You might do that for the following reasons, among others:

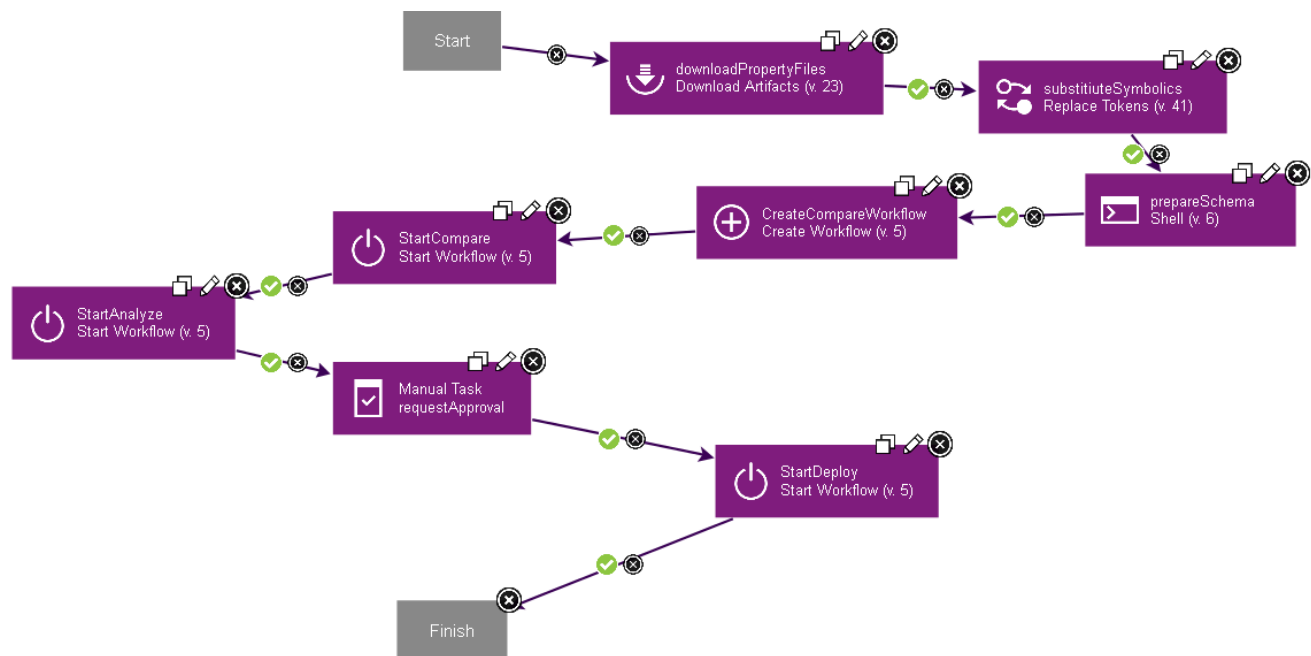
- So that Administrators can review and approve the final changes in a controlled environment
- To separate the compare from the application the final changes because the compare takes too long in a complicated environment to fit the entire process into the change window to a target environment

In such cases, the Object Compare tool workflow must be executed step-by-step, so the application deployment process can pause at the desired point.

Open the UpgradeApplicationSchemawithApproval process.

Step 3.m. Go to the Process page under the component. Click **Edit** on the right for the process UpgradeApplicationSchemawithApproval.

Figure 16. UpgradeApplicationSchemawithApproval



The first 4 steps are same as those in the UpgradeApplicationSchemawithoutApproval process. Open each step and customize the values for the properties based on your environment.

Step 3.n. Open the StartCompare step. The difference here is to run the workflow step by step instead of the entire workflow. Look at the **Step Name** and **Perform Subsequent** values. Then do the same for the StartAnalyze step, which analyzes the comparison result and determines the final changes.

Figure 17. StartCompare properties

Step 3.o. Open the Manual Task step. This is to send the analyze result to the DBA role, which was created in step 1, for review and approval. Customize the properties if you have different role name. The default value of the only added property refers to the output property of the previous StartAnalyze step, for the content of the schema changes from the analysis at the execution time.

Figure 18. Manual task properties for UpgradeApplicationSchemawithApproval

Name \*

Notification Template \*

[Add Property](#)

Name	Label	Pattern	Required	Default Value	Description	Actions
Schema change content	Schema change content		false	\$(p.StartAnalyze/zosmf.workflowJoblog)	Schema change content	<a href="#">Edit</a> <a href="#">Delete</a>

1 record - [Refresh](#) [Print](#)

Who can approve this task

for  [Remove](#)

Rows 10

Step 3.p. The StartDeploy step rolls out the final changes to a target environment if the DBA approves the changes in the previous step during the execution. The properties of the step are similar to the StartAnalyze and StartCompare steps. Open the step and review the properties. The final deployment result is shown in the output property of the step at execution time.

Next, define the target environment and setup the properties referenced by the steps in the processes.

#### Step 4. Add target environments

Step 4.a. Go to the Resources page, click **Create Top-Level Group**, and enter a name.

Figure 19. Add resource group

Create Resource ? ✕

Name \*

Include Agents Automatically ☐

Description

Teams +

[Save](#) [Cancel](#)

Step 4.b. Open the resource group, click **Create > Agent** to add an agent from the list, enter a name and save. The tutorial discusses teams later.

Figure 20. Add an agent

**Create Resource** ? ✕

---

Agent

Description

Inherit Teams From Parent

☒

Teams

DBATeam

SECADMTeam

UCDUserTeam

Default Impersonation

☐

Default impersonation can be configured here. Any steps which do not specify their own impersonation settings will fall back to the settings provided here.

Save

Cancel

Step 4.c. Click to open the added agent, then click **Create** and choose **Component** to add the component that you choose in Step 2 from the list.

Step 4.d. Go to the Configuration panel under the agent, click **Resource Properties** on the left, and add the properties but customize the values based on your environment as the below figure. These properties are used to run the SQL statements in the last step of the process by the JDBC plugin, and the plugin uses the RACF Passticket, so no password is required.

Figure 21. Resource properties

Basic Settings

Resource Properties

## Resource Properties

Version 11 of 11

◀◀ ◀ ▶ ▶▶

Add Property

Batch Edit

Name	Value
<input type="text"/>	<input type="text"/>
connectionURL	jdbc:db2://dtec954.vmec.svl.ibm.com:446/STLEC1
connectionUser	SYSADM
jdbcDriver	com.ibm.db2.jcc.DB2Driver
jdbcJAR	/usr/lpp/db2/devbase_jdbc/classes/db2jcc4.jar
jesHost	dtec954.vmec.svl.ibm.com
jesLibPath	/usr/lib
jesPort	6715
jesPTID	SYEC1DB2
jesRACF	/usr/include/java_classes/IRRRacf.jar
jesUser	SYSADM

`connectionURL` is the jdbc connection URL for the target environment

`connectUser` is the user name for the JDBC connection, and the name must be granted by RACF Passticket for JDBC ahead

`jdbcDriver` is the driver name for DB2 for z/OS

`jdbcJAR` is the full path name pointing to the db2 jcc jar file on the USS of the target environment

`jesHost`, `jesLibPath`, `jesPort`, `jesPTID`, `jesRACF` and `jesUser` are RACF Passticket values for JES. However, they are also shared by the RACF Passticket for JDBC. The RACF Passticket for JES allows UCD to run JCL statements without requiring a password stored outside. Customize the values based on your environment

Step 4.e. Go to the Applications page, open the ApplicationDeployment application. Two environments are already defined by default, you can choose to rename the environment or create your own by clicking **Create Environment**.

After you open a created environment, click **Add Base Resources** and select the agent and component that you added in previous steps.

Figure 22. Add base resources

Add Base Resources
Select All...
Actions...
Show

	<input type="checkbox"/>	Name
--	--------------------------	------

Step 4.f. Go to the Configuration page under the environment, click **Environment Properties** from the left, and customize the properties based on your environment. Refer to the following figure.

**DBSCHEMAFILE** is a file name for the extracted or downloaded DDL under USS

**SYSTEMID** is the id of the target environment

**WORKFLOWSERVER** is the hostname of the target environment where z/OSMF is running

**WORKFLOWUSER** is the user name which will be the owner of the workflow to be created on z/OSMF

**aocCMPProfDataset** is a dataset keeping the profile to Object Compare Tools workflow, which will be used in application deployment process. Add this property if you may also want to run that process against the environment, otherwise, skip this one

**aocCMPProfFile** is the name of the property file to keep your own values of the profile to Object Compare Tools workflow. Add this property if you may also want to run that process against the environment, otherwise, skip this one

**aocCMVars** is the name of the property file to keep the values of the parameters to Object Compare Tools workflow. Add this property if you may also want to run that process against the environment, otherwise, skip this one

**aocGenVars** is the name of the property file to keep the values of the parameters to Administration Tools workflow

**aocMaskDataset** is a dataset keeping the mask definition to Object Compare Tools workflow. Add this property if you may also want to run that process against the environment, otherwise, skip this one

**aocMaskDefFile** is the name of the property file to keep the values of the mask parameters to Object Compare Tools workflow. Add this property if you may also want to run that process against the environment, otherwise, skip this one

**aocworkflowDirectory** is the full path of the folder where your Administration Tools and Object Compare Tools workflows locate after APAR PI67731

**ddlOutputDataset** is the dataset keeping the extracted data definition statement. content by the Administration Tools workflow, and **sourceDDLDataset** is the same

dataset without the high qualifier in this tutorial, but they could be different. In this tutorial, the high-level qualifier uses WORKFLOWUSER as default. Customize the value correspondingly based on your environment

**srcDirectory** is the path for the downloaded artifacts and working files while the processes are being executed. Ensure that both UCD and z/OSMF server have the privilege to read the folder and files under the folder on USS

**workflowPwd** is the password for the WORKFLOWUSER on the z/OSMF server

Refer to the APAR PI67731 for the detail description for the parameters, profile properties, and mask properties to Administration and Object Compare Tools workflows.

The default environment parameter values, **profile file**, **mask file**, **aocGenVars** and **aocCMVars** files are provided as the sample for reference along with the template package. Be sure to customize the values based on your environment.

*Figure 23. Environment properties*

DBSCHEMAFILE

All Components

[Split Values Per Component](#)

SYSTEMID

All Components

[Split Values Per Component](#)

WORKFLOWSERVER

All Components

[Split Values Per Component](#)

WORKFLOWUSER

All Components

[Split Values Per Component](#)

aocCMProfDataset

All Components

[Split Values Per Component](#)

aocCMProfFile

All Components

[Split Values Per Component](#)

aocCMVars

All Components

[Split Values Per Component](#)

aocGenVars

All Components

[Split Values Per Component](#)

aocMaskDataset

All Components

[Split Values Per Component](#)

aocMaskDefFile

All Components

[Split Values Per Component](#)

aocworkflowDirectory

All Components

[Split Values Per Component](#)

ddlOutputDataset

All Components

[Split Values Per Component](#)

sourceDDLdataset

All Components

[Split Values Per Component](#)

srcDirectory

All Components

[Split Values Per Component](#)

workflowPwd

## Step 5. Setup deployment style and notification

The deployment can be scheduled to run automatically. The following steps show 2 optional ways on automating the deployment.

Option 1. Go to the Applications page, click **ApplicationDeployment**, then click **Request Process** beside the created environment. You can check **Schedule Deployment**, to set the date and time to kick off the deployment at that time; you can also make the deployment happen recursively by checking **Make Recurring** if there is a need to provision or deploy application change on a regular basis.

Figure 24. Run process on an environment

Run Process on IntegrationTestDeploymentEnv ✕

---

Only Changed Versions ☒

Process\* 

UTDeployRTCProcess ▾

Select a snapshot, or choose versions for individual components.

Snapshot 

▾

Schedule Deployment? ☒

Date\* 

8/31/2016 ▾

Time\* 

4:48 PM ▾

Make Recurring ☒

Pattern\* 

Monthly ▾

Description

---

Submit

Cancel

Option 2. Go to the Components page, click the component that you configured in previous steps. Then go to the Configuration panel under the component, and select **Basic Settings**.

Figure 25. Basic settings

The screenshot shows a configuration interface for UCD server settings. It includes several sections with checkboxes and dropdown menus. The 'Import Versions Automatically' section has a checked checkbox. The 'Copy to CodeStation' section also has a checked checkbox. The 'Default Version Type' section has a dropdown menu set to 'Full'. Below this are three radio buttons: 'Use the system's default version import agent/tag.' (selected), 'Import new component versions using a single agent.', and 'Import new component versions using any agent with the specified tag.'. The 'Cleanup Configuration' section has a checked checkbox for 'Inherit Cleanup Settings'. Below this is a checked checkbox for 'Run Process after a Version is Created'. The 'Application Process' dropdown is set to 'ApplicationDeployment - UTDeployRTCPProcess'. The 'Environment' dropdown is set to 'ApplicationDeployment - IntegrationTestDeployme'.

Import Versions Automatically ☒

Copy to CodeStation ☒

Default Version Type \* Full

☒ Use the system's default version import agent/tag.  
☐ Import new component versions using a single agent.  
☐ Import new component versions using any agent with the specified tag.

Cleanup Configuration

Inherit Cleanup Settings ☒

Run Process after a Version is Created ☒

Application Process \* ApplicationDeployment - UTDeployRTCPProcess

Environment \* ApplicationDeployment - IntegrationTestDeployme

Check **Import Versions Automatically** to let UCD server import a newly checked in version (a version after build succeeds) automatically using the SCM information that you set in Step 2. This is useful when developers want to automate the deployment of any application and schema changes right after they check in the code into SCM and the code is built successfully, with the integration of UCD, SCM and Build systems.

You can also check **Run Process after a Version is Created** to kick off the chosen process immediately after the new version is imported.

Notification is a preferable way to know when a process is kicked off, whether it succeeds or fails, when a process is ended, whether a review or approval is needed or pending, and so forth.

In order to set notification, go to the Settings page if you are administrator of UCD server. Otherwise, you might need the administrator to grant you the related privileges. Click **Notification Schemes** on the right, then either create a new notification scheme, or modify an existing one. Click the notification scheme you want to customize, then click **Add Notification Entry** to add an entry.

*Figure 26. Add notification entry*

## Add Notification Entry



---

Role *	UCDUsers ▼
Event Type *	Process Started ▼
Target *	Application ▼
Type	Standard Application ▼
Template Name *	ProcessRequestStarted ▼

---

Save

Cancel

The users who are assigned to the role are notified when the selected event occurs. Add as many as you want to get the proper roles notified for each event.

You can choose to use the default notification template in UCD or to create your own notification template. Refer to the UCD knowledge center to find out more on creating or customizing notification template.

We have been talking about the role and team in the previous sections. To manage roles and teams in UCD, go to the Settings page and choose corresponding entries under Security to add, edit, or remove users, roles, or teams in UCD. Refer to the UCD knowledge center to find out more on security administration tasks.

### Step 6. Validate the processes

The sample application processes are already created in the template and are ready to validate the entire flow. You can go to "Processes" panel under the application to review and customize.

You are ready now to validate what you have configured so far. Choose **Request Process** against an environment from the Environments panel under an application, and select and submit the process.

UCD brings you to the page which displays the execution status of the process. Wait for the process complete and check the status.

Everything in green means that you have done a great job!

## Part II. Bind Package Process

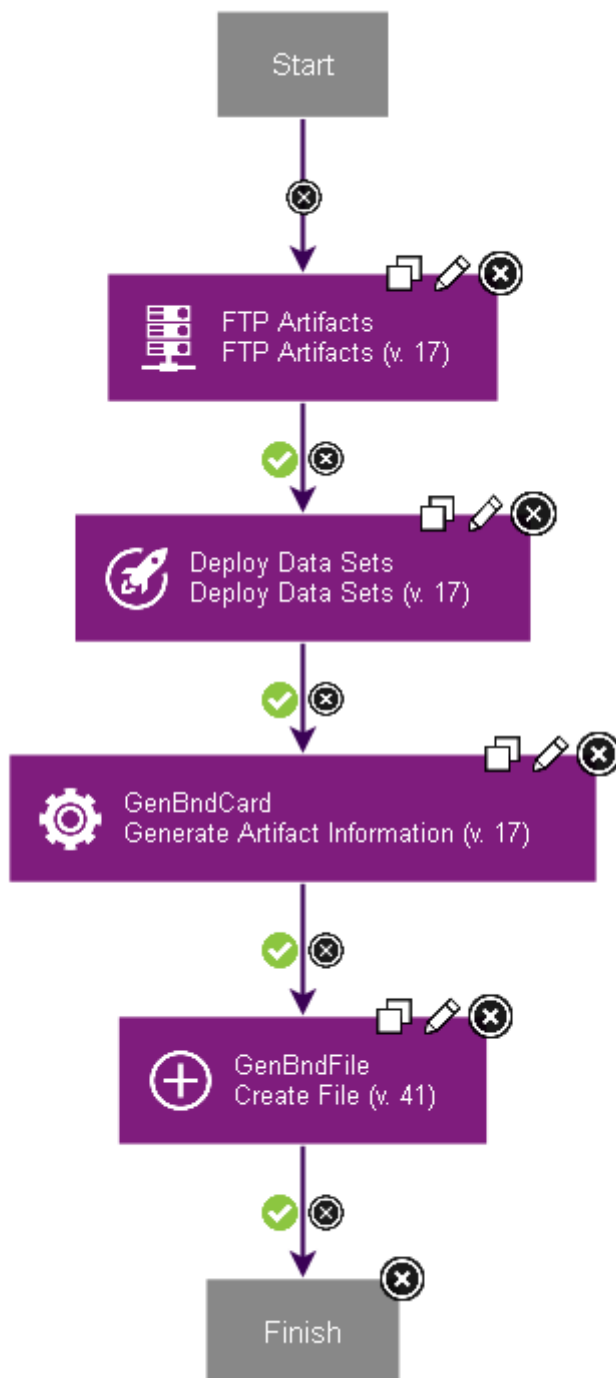
The template imported in Part I also contains the application, components, processes and all artifacts to be discussed Part II, besides, the SCM system configuration for the components to be discussed in Part II is same as that in Part I, so we are going to skip the import and SCM setup steps. Refer to the first 2 steps in Part I if you need.

The deployment style and notification setup for the solution in Part II is also similar like that in Part I, so we are also going to skip this part. Refer to the Step 5 in Part I if you want to configure the components of Part II in the same way.

Step 1. Implement the process which prepare the DBRMs to be bound and the related BIND job based on the BIND cards

Step 1.a. Click the DeployDemoPrep process on the Processes panel, and click **RDProcess** to review and customize.

*Figure 27. RDProcess process*



Step 1.b. Open the FTP Artifacts step, and customize and save the properties of the step as shown in the following figure.

Figure 28. Edit Properties of FTP Artifacts

Edit Properties

Name \*

FTP Artifacts

Directory Offset \*

.

Working Directory

Post Processing Script

Step Default

New

Precondition

Use Impersonation

☐

Show Hidden Properties

☒

Host Name \*

\$(p:component/repositoryHost)

User Name \*

\$(p:component/repositoryUser)

Password \*

\$(p:component/repositoryPwd)

Repository \*

\$(p:component/repositoryDir)

Version Name \*

\$(p:version.name)

Component Name \*

\$(p:component.name)

Repository Type \*

\$(p?:version/ucd.repository.type)

OK

Cancel

Check **Show Hidden Properties** at the bottom to see additional properties for defining the related information of the repository which is used to keep the DBRMs. You can use the names of component properties as the values for the inputs here, and define the real values in the component properties later.

Step 1.c. Open the Deploy Data Sets step, and customize and save the properties as shown in the following figure.

Figure 29. Edit Properties of Deploy Data Sets

**Edit Properties** ✕

---

**Name \***

**PDS Mapping**

**Check Access** ☒

**Allow Creating Data Set**

**Working Directory**

**Post Processing Script**  ▼

**New**

**Precondition**

**Use Impersonation** ☐

**Show Hidden Properties** ☒

**Deployment Base Path \***

**Version Name \***

**Component Name \***

---

**OK** **Cancel**

You use an environment property for the **PDS Mapping** because the target dataset name might be different for each individual environment. You define the environment property later. **Allow Creating Data Set** is set to TRUE to enable IBM UrbanCode Deploy to create the target PDS dataset with the default settings, if the dataset is not available. Set the value to FALSE if your shop doesn't want this behavior. However,

then you need to ensure that the target PDS dataset is created on the target environment before you kick off the deployment process.

Step 1.d. Open the step "GenBndCard", customize and save the properties as shown in the following figure:

Figure 30. Edit Properties of Generate Artifact Information

## Edit Properties



Name \*

GenBndCard

For Each \*

Member

Source Data Set Name Filter

Target Data Set Name Filter

Member Name Filter

Deploy Type Filter

DBRM

Custom Properties Filter

Template \*

```
BIND PACKAGE(@COLLID@  
MEMBER(${member}) +  
ACTION(@ACTION@  
ISOLATION(@ISOLATION@ +  
RELEASE(COMMIT) ENCODING(EBCDIC) +  
LIBRARY('${dataset})
```

Working Directory

Post Processing Script

Step Default

New

Precondition

Use Impersonation

☐

Show Hidden Properties

☒

Deployment Base Path

\${BUZ\_DEPLOY\_BASE}

Version Name

\${p.version.name}

Component Name

\${p.component.name}

OK

Cancel

This step iterates all of the PDS dataset members and generates the text, BIND card, in this sample with the **Template** input . For **Template** in Figure 8, `${member}` is the symbol representing each PDS dataset member; the string quoted with the character '@', like `@COLLID@`, is the symbol that is substituted with the environment-specific value later during main deployment process. The character '@' is the default quotation mark for symbolic substitution in IBM UrbanCode Deploy. You can use other special characters if '@' is not a good choice for your shop.

In this sample, you put the BIND card template directly in the step. However, you can also use the following **Template** value if your shop has the BIND cards stored in one or multiple files:

```
@${member}@ -  
LIBRARY('${dataset}')
```

Here `${member}` is a property name to be replaced later with the real value, in the symbolic replacement step. While the real BIND card is kept in one or multiple text property files. The property key-value pairs are defined in the following multi-line format (taking the DSNADMCD DBRM member as a sample here):

```
DSNADMCD = \  
    BIND PACKAGE(DB2OSC) MEMBER(DSNADMCD) - \n\  
    ACTION(REPLACE) ISOLATION(CS) - \n\  
    RELEASE(COMMIT) ENCODING(EBCDIC)
```

Thus, you can have different combination of BIND options for different DBRM members.

The symbols are very useful for deploying the application to different environments, especially in multiple tenant environments, where you might want to create the data objects under different schemas and to bind the packages with different qualifiers under different collection ids for different end users, so that different end users can share the same subsystem without interference.

You can use symbols for anything, like syntax options, parameters and so on, whichever shows up in your artifacts, as long as you have the real value to replace that later.

Step 1.e. Open the GenBndFile step under the FileUtils plugin, and define the properties as shown in the following figure:

Figure 31. Edit Properties of Create File

✕

Edit Properties

Name \*

GenBndFile

File Name \*

\$(p:component/bind)

Contents

//BINDPKG JOB  
'USER=\$\$USER',\$\$USER',CLASS=G,PRTY=11,  
// MSGCLASS=H,MSGLEVEL=  
(1,1),USER=@JCLUSER@,  
// PASSWORD=CODESHOP,REGION=4096K  
/\*ROUTE PRINT @JCLPRINT@

Overwrite if exists

☒

Working Directory

\$(p:environment/srcDirectory)

Post Processing Script

Step Default

New

Precondition

Use Impersonation

☐

Show Hidden Properties

☐

OK

Cancel

Use a component property for the name of the BIND job because this can be same for all environments. You define this property later. For **Contents**, enter a template JOB header with symbols which will be substituted in the main deployment process, and use the BIND cards, `$(GenBndCard/tex)`, generated from the previous step. **Working Directory** defines a directory to contain the BIND job. This will be the same directory for other artifacts in the main deployment process. Note that you can use the same method explained in the previous step to set a property for job header to generate different headers for different environments, if necessary.

Step 1.f. Define the required properties for the AppDeployPackagePrep component .

Click the AppDeployPackagePrep component. On the Configuration panel, and switch to the Component Properties tab.

Customize the properties as shown in the following figure.

Figure 32. Component properties of the AppDeployPackagePrep component

**Component Properties**

Version 9 of 9

◀◀ ◀ ▶ ▶▶

**Add Property** **Batch Edit**

Name	Value
<input type="text"/>	<input type="text"/>
bind	bind.jcl
repositoryDir	/u/oeusr05/opt/ucdagent/var/repository
repositoryHost	labec431.vmec.swl.ibm.com
repositoryPwd	****
repositoryUser	sysadm

Go to the Environment Property Definitions page, and customize properties as shown in the following figure. This is an optional step.

Figure 33. Environment properties of the AppDeployPackagePrep component

**Environment Property Definitions**

Define properties here to be given values on each environment the component is used in.

**Add Property**

Version 3 of 3

◀ ▶ ▶▶

Name	Label	Pattern	Required	Default Value
pdsMapping	pdsMapping		false	DSNC10.SDSNDBRM,KREN.CNTLDBRM
srcDirectory	srcDirectory		false	/u/oeusr05/testsrc

Now you probably notice an interesting thing.

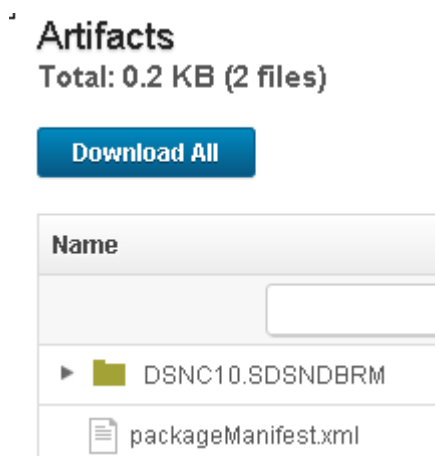
As mentioned earlier, these 2 symbols are at the environment level and can have different values for each target environment in the deployment. Why define them in the component? You can trust that this is not a mistake. This step actually defines the default values for these environment properties at the component level. In other words, if you don't provide the real values for a certain target environment, then the deployment process uses the default values that you provide here. If you define the

real values for each target environment as in step 5, the values given in the step 5 overwrite the default values defined here during the deployment. This gives you one way to share some common settings among some environments. Of course, you don't have to do this if this is not the case you want.

Step 1.g. Import the DBRMs into the AppDeployPackagePrep component with BUZTOOL and the sample job, SBUZSAMP(BUZRJCL), supplied by IBM UrbanCode Deploy. You can customize and run the sample job to get the DBRMs from the source dataset into the component in IBM UrbanCode Deploy. The packageManifest.xml in the template shows how to compile the list of the required DBRMs as the input list to be imported. You could find more detail of BUZTOOL on [http://www-01.ibm.com/support/knowledgecenter/api/content/SS4GSP\\_6.1.3/com.ibm.udeploy.doc/topics/zos\\_runtools.html?locale=en](http://www-01.ibm.com/support/knowledgecenter/api/content/SS4GSP_6.1.3/com.ibm.udeploy.doc/topics/zos_runtools.html?locale=en).

After it is done, you see the imported artifacts listed as shown in the following figure:

Figure 34. Artifact list of the DeployDemoPrep step

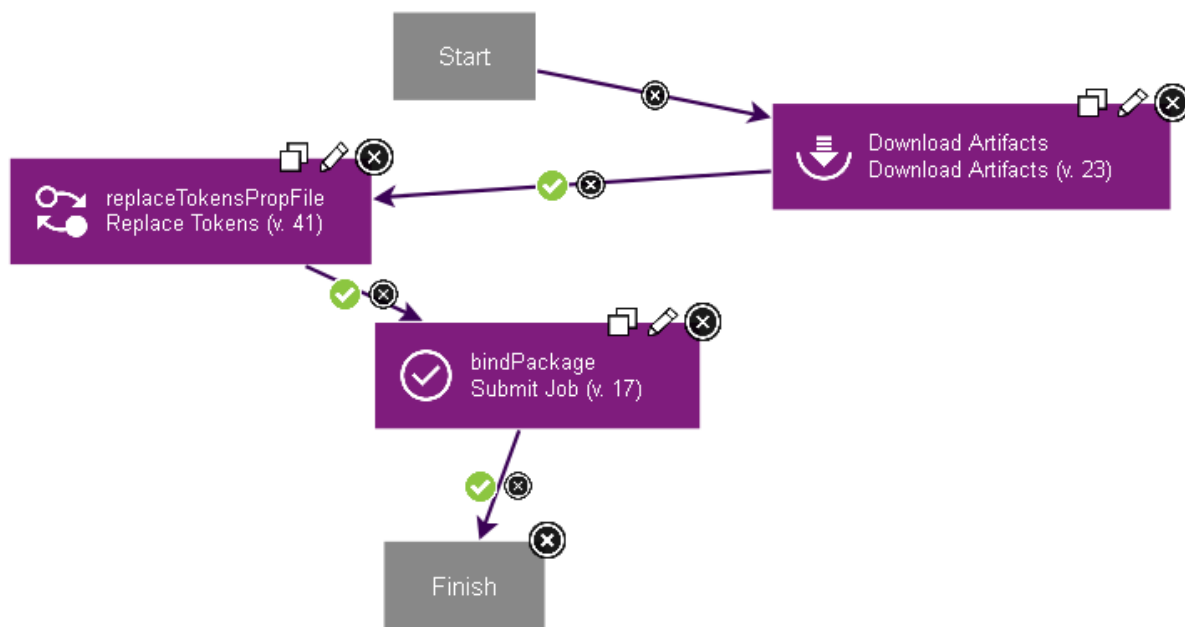


The DeployDemoPrep component is now ready. If you need to deal with load modules, the process is similar and simpler, and you can skip the BIND card and the step to generate the BIND job.

## Step 2: build the main process that binds the packages

Step 2.a. Click the component "AppDeployPackageRTC" or "AppDeployPackageGIT". On the Processes panel, and click the sample uses "BindPackages".

Figure 35. The BindPackages process



Step 2.b. Open the step “Download Artifacts”, customize and save the properties as shown in the Figure 36 if you use AppDeployPackageRTC, or the Figure 37 if you use AppDeployPackageGIT.

*Figure 36. Edit properties of the Download Artifacts step, for RTC*

## Edit Properties

---

<b>Name *</b>	<input type="text" value="Download Artifacts"/>
<b>Directory Offset *</b>	<input type="text" value="."/>
<b>Artifact Directory Offset</b>	<input type="text" value="\${p:component/projectname}"/>
<b>Includes *</b>	<div><div>**/*</div><div></div></div>
<b>Excludes</b>	<div><div></div><div></div></div>
<b>Sync Mode</b>	<div>none ▾</div>
<b>Full Verification</b>	<input checked="" type="checkbox"/>
<b>Set File Execute Bits</b>	<input type="checkbox"/>
<b>Verify File Integrity</b>	<input type="checkbox"/>
<b>Charset</b>	<input type="text"/>
<b>Working Directory</b>	<input type="text" value="\${p:environment/srcDirectory}"/>
<b>Post Processing Script</b>	<div><div>Step Default ▾</div><div>New</div></div>

Figure 37. Edit properties of the Download Artifacts step, for GIT

## Edit Properties



**Name \***

**Directory Offset \***

**Artifact Directory Offset**

**Includes \***

\*\*/\*

**Excludes**

**Sync Mode**

**Full Verification** ☒

**Set File Execute Bits** ☐

**Verify File Integrity** ☐

**Charset**

**Working Directory**

**Post Processing Script**

Step Default

**New**

**Precondition**

**Use Impersonation** ☐

**Show Hidden Properties** ☐

OK

Cancel

Use the same working directory as the process of component “DeployDemoPrep” to put all the deployable artifacts under the same folder on the target environment.

Step 2.c. Open the replaceTokensPropFile step and customize and save the properties as shown in the following figure.

Figure 38. Edit properties of the replaceTokensPropFile step

## Edit Properties

---

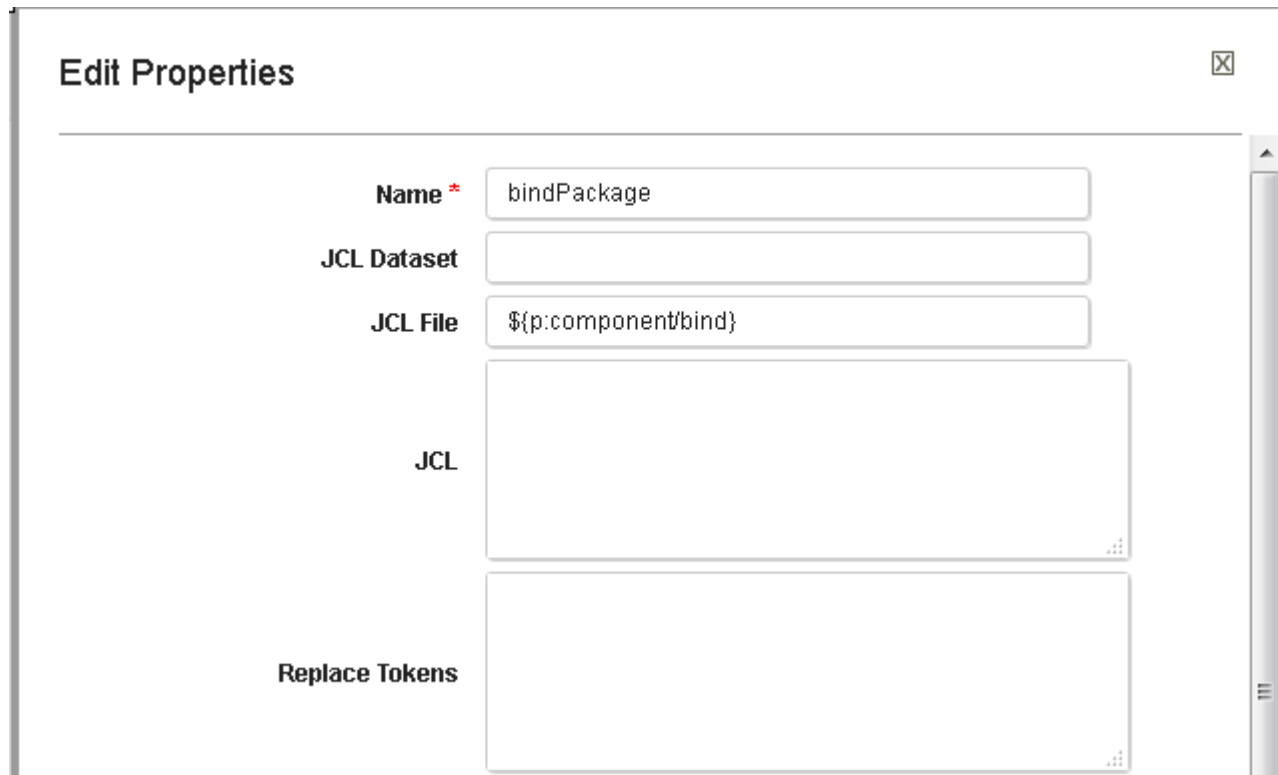
Name *	replaceTokensPropFile
Directory Offset	.
Include Files *	*.jcl
Exclude Files	
Start Token Delimiter	@
End Token Delimiter	@
Property Prefix	
Property File Name *	\${p:environment/symbolicReplacement}
Property List	
Explicit Tokens	
Working Directory	\${p:environment/srcDirectory}

This step is to substitute the symbols including BIND card or Job header if applicable in the bind job which is generated in step 1. Enter \*.jcl for **Include Files**. Use **Start**

**Token Delimiter** and **End Token Delimiter** to define a token delimiter for your shop. The default delimiter is the '@' character. In this sample, use the text property files to feed the real environment values into the symbols in all the files which contain the symbols. The working directory is pointing to the same folder where that contains all the artifacts on the target environment.

Step 2.d. Open the bindPackage step and customize and save the properties as shown in the following figure.

Figure 39. Edit properties of the bindPackage step



The screenshot shows a dialog box titled "Edit Properties" with a close button in the top right corner. The dialog contains several input fields and checkboxes:

- Name \***: A text field containing "bindPackage".
- JCL Dataset**: An empty text field.
- JCL File**: A text field containing "\${p:component/bind}".
- JCL**: A large, empty text area.
- Replace Tokens**: A checkbox that is currently unchecked.

Set the bind job in one component property and use that for **JCL File**. Check **Show Hidden Properties**, and set other fields as shown in the following figure.

The method described here can also be used to launch other DB2 commands or utilities like RUNSTATS, REORG, CHECK DATA, and others.

Figure 40. Hidden properties of the bindPackage step

Show Hidden Properties ☒

Host Name *	<input data-bbox="472 201 1073 254" type="text" value="\${p:resource/jesHost}"/>
Job Monitor Port *	<input data-bbox="472 264 1073 317" type="text" value="\${p:resource/jesPort}"/>
User Name *	<input data-bbox="472 327 1073 380" type="text" value="\${p:resource/jesUser}"/>
Password	<input data-bbox="472 390 1073 443" type="password"/>
Use Passticket	<input checked="" type="checkbox"/>
IRRRacf.jar File *	<input data-bbox="472 516 1073 569" type="text" value="\${p:resource/jesRACF}"/>

RACF Passticket is supported to run JCL too, so you don't have to store the password in the process. The real values of other fields are defined at the resource level, like as JDBC properties.

For the details for RACF Passticket support in the Submit Job of the zOS utility plugin, see [z/OS Utility plug-in](#). You need to ensure that the RACF Passticket is configured on the target subsystem successfully for Job Monitor which is used by the plugin here, before you run your deployment process at the last step.

Step 2.e. Customize the required component properties. Click the component in the component list, and go to the Configuration panel. Switch to the Component Properties page. and customize and save the properties.

### Step 3: Add the environments and components

Step 3.a. Open the created resource in Part I Step 4, and add the components DeployDemoPrep and AppDeployPackageRTC or AppDeployPackageGIT from the list.

Step 3.b. Open the created environment in Part I Step 4 or create a new environment by following the same steps in Part I Step. Customize and save the properties.

The `bindsym.prop` text file defines the symbols and the environment specific values for this environment. The file is in a simple format with a list of key-value pairs, and each key-value pair takes one line. You can refer to the file supplied in the template package as a sample to compile your own.

The property file provides the flexibility for you to add any new symbols or values, without changing whatever you defined in the IBM UrbanCode Deploy. The property file can be upgraded and managed along with all other artifacts in the source control management solution. The alternative is to define the symbols as the properties on each target environment Then you must modify the configuration to add, update, or remove a property in IBM UrbanCode Deploy every time that you need to make a change to a symbol.

Another advantage of using the property file is that you can easily share the same set of real values among various target environments. However, if you define the symbols as the properties directly in IBM UrbanCode Deploy, you might have to duplicate the definition in multiple places.

As you already learned from the preceding steps, you use symbols in the BIND card and utility jobs, which enable you to easily add new syntax from future DB2 releases. You just check in the new artifacts and import them as a new version into the component, and run the deployment process.

Everything is defined well now, and you are ready to deploy.

#### Step 4: Validate the entire bind package process

The sample application processes are already created in the template and are ready to validate the entire flow. You can go to the Processes panel under the application to review and customize.

On the Applications page, click **Request Process**, which is a small one beside your target environment, against the target environment. Choose the corresponding process and version for each component, then submit the request.

A page opens where you can monitor the results of the deployment. Now it is time to grab a cup of coffee, or another!

You can see the process run successfully after you return. Congratulations!

## Deploy the application with batch command

---

IBM UrbanCode Deploy supports REST APIs and CLI commands, which enable you to create all the required elements of the deployment process and to run the deployment process.

You can build a batch script, or a job, or even a program to implement the entire or part of work that is described above with REST APIs or CLI commands based on your needs, for example, you can use a batch script to add a target environment and to deploy the application to that instead of going through the GUI interface described above. The batch script or job can be integrated in or launched by the specific procedure in your shop.

This can also be integrated with other provisioning solution in a cloud computing environment. For example, a developer in your shop might request a DB2 for z/OS

subsystem for unit testing of certain applications on demand, the integrated solution can build an ad hoc subsystem with the applications deployed per developer's specific request. The environment can be de-provisioned after the developer's work is done. Furthermore, the entire solution can be made as a self-service.

Below is a sample command to launch an application process via REST call.

```
curl -k -u <urbancode deploy userid>:<urbancode deploy user pwd>  
https://plxeditor.usca.ibm.com:8443/cli/applicationProcessRequest/request -  
X PUT -d @provisionlike.json
```

## Summary

---

In this tutorial, you learned how to build a solution to provision schema on an environment like the one with golden copy of the schema, to build a solution to deploy the schema changes to environments with previous versions, to build a solution to deploy packages (bind or rebind) against environments and how to launch the DB2 utilities and commands with RACF PassTicket support in the sample.

The tutorial also elaborated the usage with REST APIs and CLI commands, and how to deploy the application on an ad hoc environment with REST APIs.

## Resources

---

- Find more useful resources on [UrbanCode Deploy](#).
- Find more plugins on [UrbanCode Deploy Plugins](#).